FONDATION WIENER-ANSPACH

European Research Council
Established by the European Commission

Incremental Techniques for Large-Scale Dynamic Query Processing

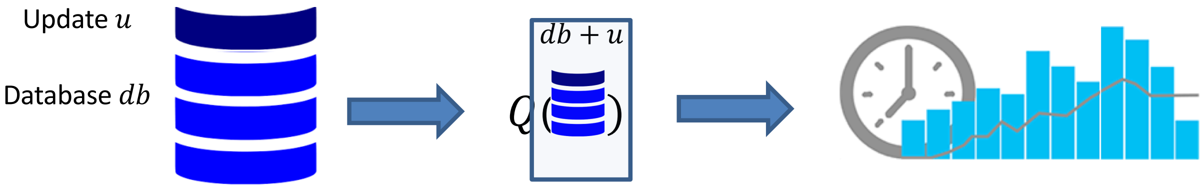
Tutorial

Part 1

Iman Elghandour[1] Ahmet Kara[2] Dan Olteanu[2] Stijn Vansummeren[1]

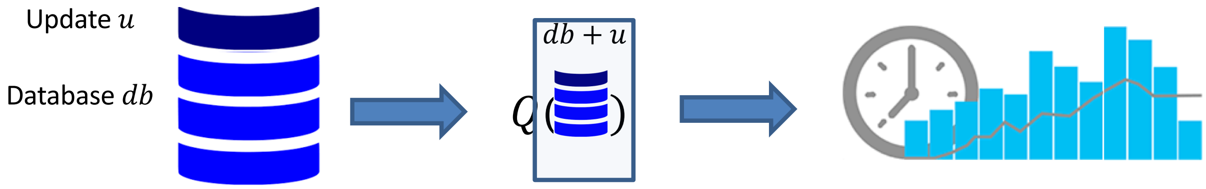
[1] ULB UNIVERSITÉ LIBRE DE BRUXELLES

[2] UNIVERSITY OF OXFORD



Dynamic query evaluation

Avoid full recomputation – compute incrementally!



Update u

Database db

$db + u$

$Q($ $)$

Application Scenarios:

- Real-time monitoring
- Internet of things

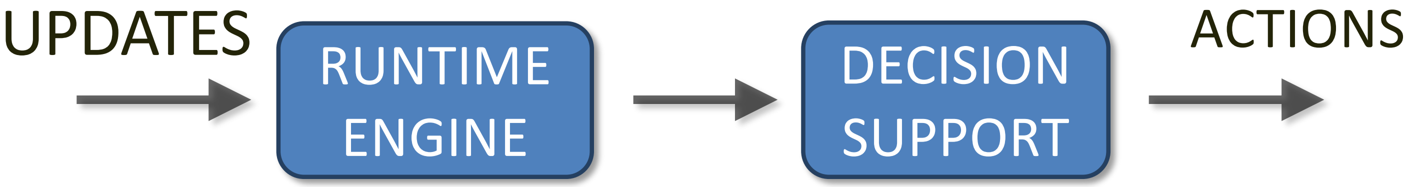
- Knowledge base construction
- Online machine learning

Real-time monitoring

Web Analytics

Sensor Networks

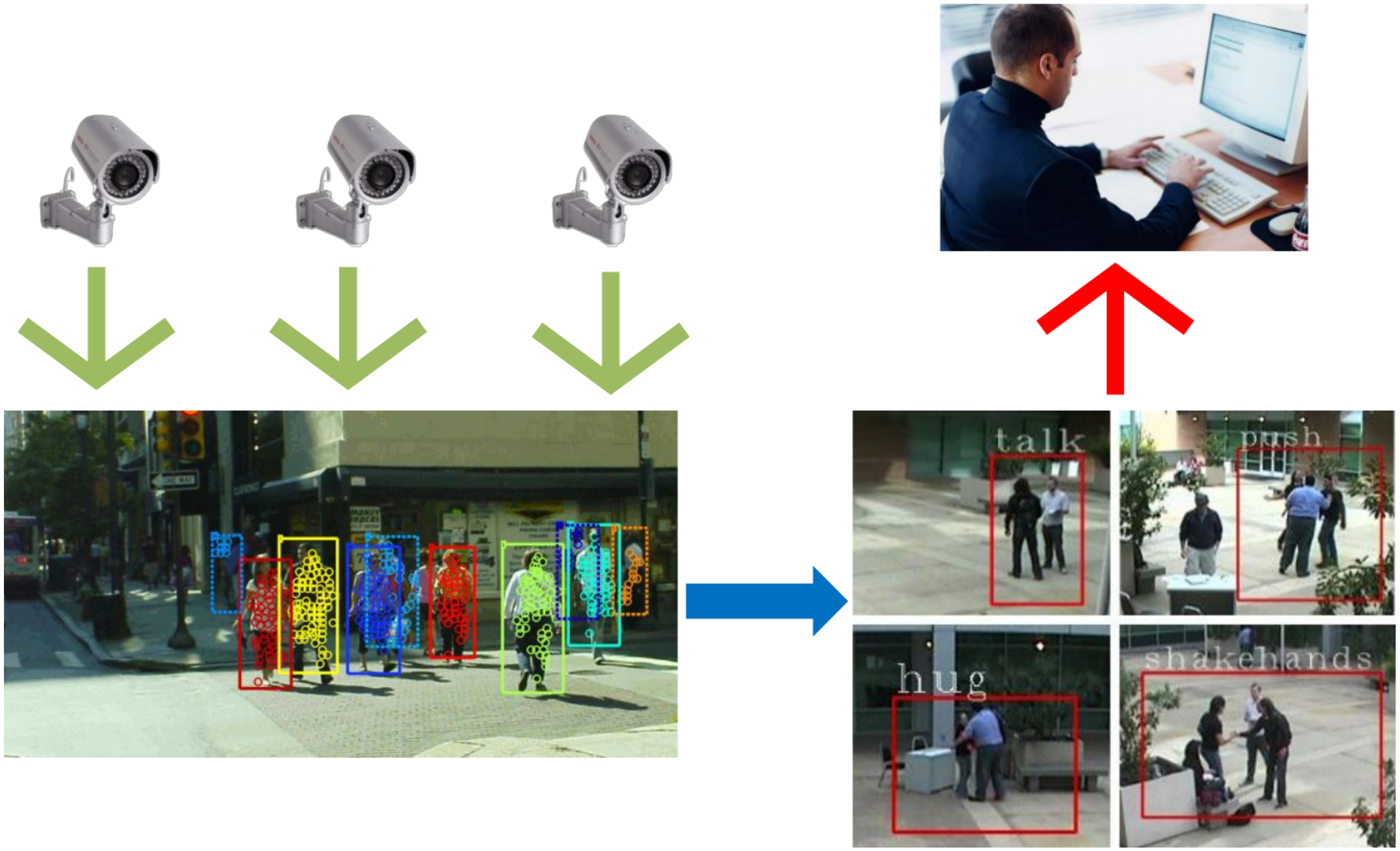
Cloud Monitoring



Continuously arriving data

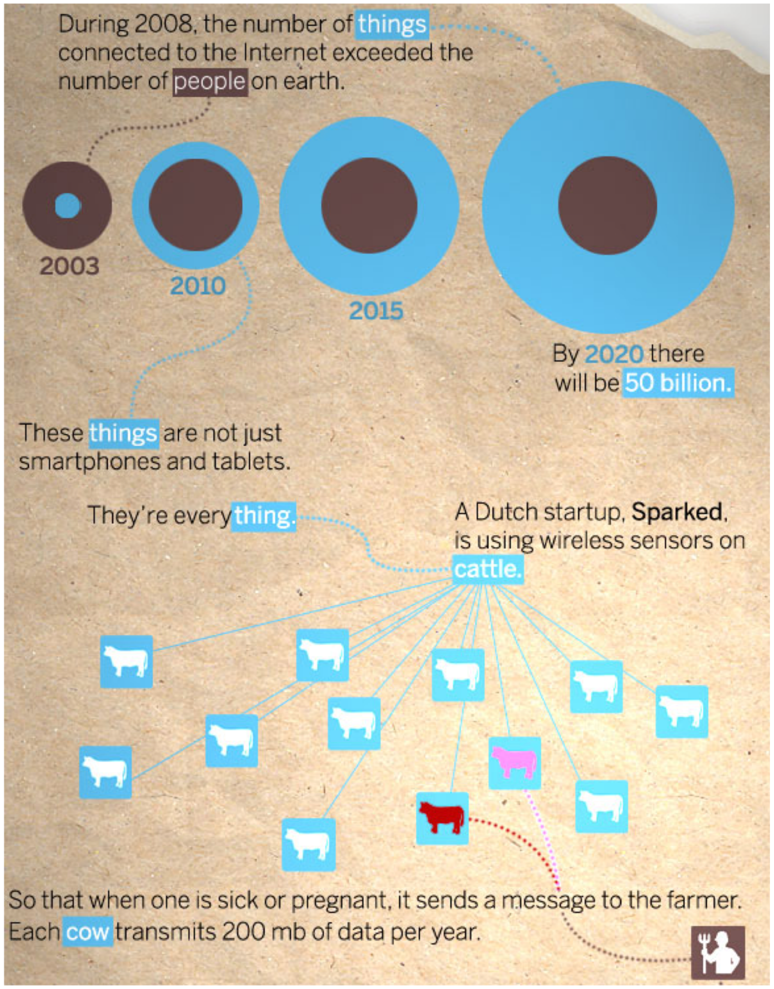
Continuously evaluated views

Complex Event Recognition



A. Artikis. Complex Event Recognition. European Conference on Artificial Intelligence (ECAI), 2016.

Internet of things



During 2008, the number of things connected to the Internet exceeded the number of people on earth.

By 2020 there will be 50 billion.

These things are not just smartphones and tablets.

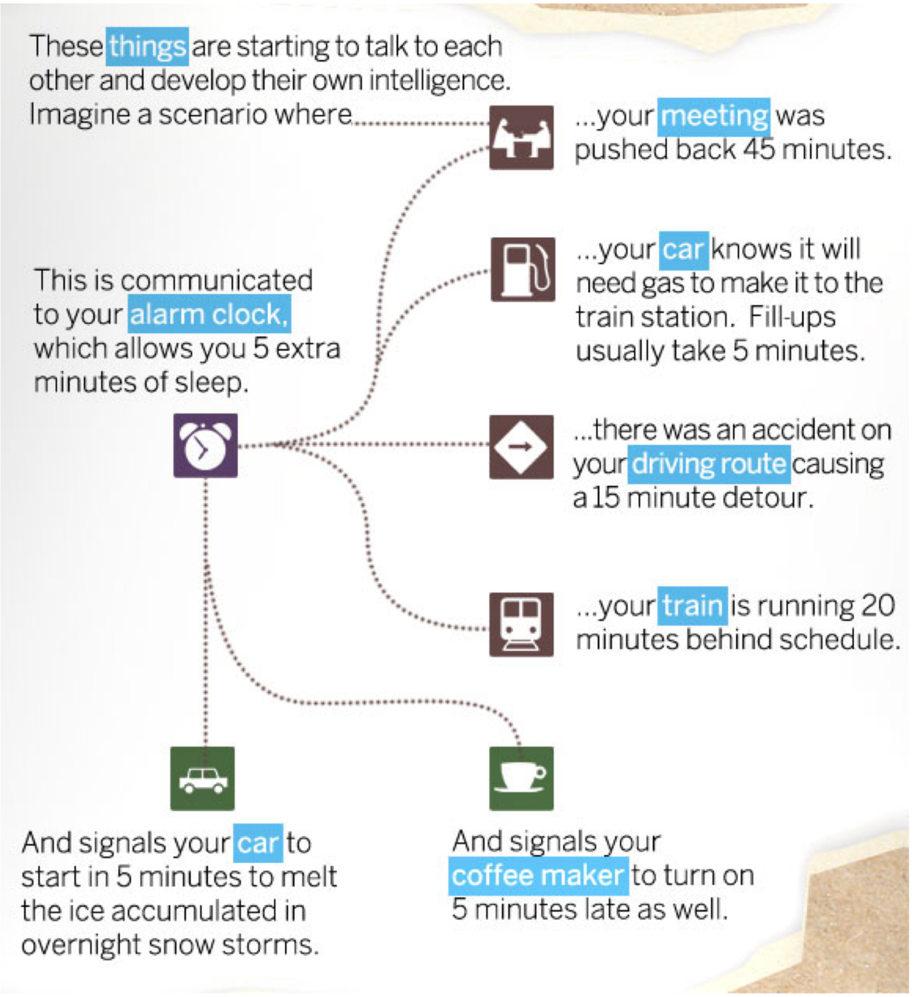
They're everything.

A Dutch startup, **Sparked**, is using wireless sensors on cattle.

So that when one is sick or pregnant, it sends a message to the farmer. Each cow transmits 200 mb of data per year.

Source: https://blogs.cisco.com/diversity/the-internet-of-things-infographic

Internet of things



These things are starting to talk to each other and develop their own intelligence. Imagine a scenario where...

...your meeting was pushed back 45 minutes.

...your car knows it will need gas to make it to the train station. Fill-ups usually take 5 minutes.

...there was an accident on your driving route causing a 15 minute detour.

...your train is running 20 minutes behind schedule.

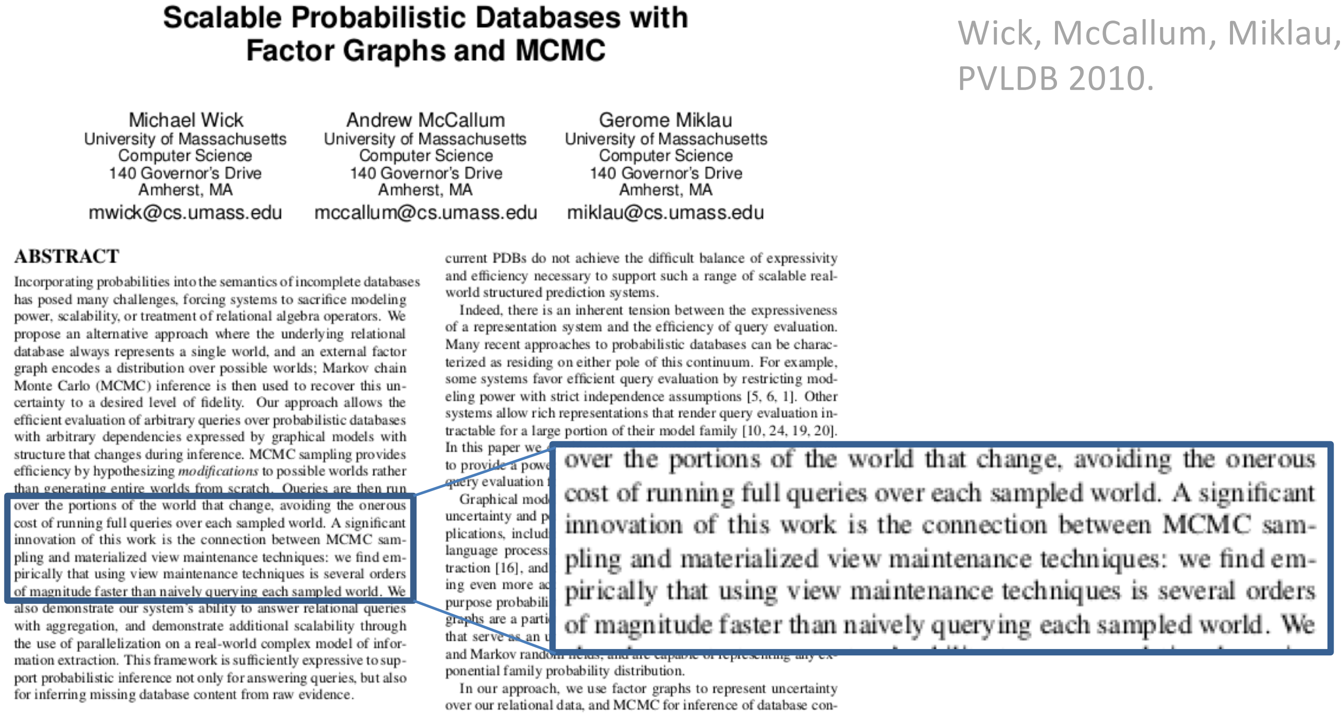
This is communicated to your alarm clock, which allows you 5 extra minutes of sleep.

And signals your car to start in 5 minutes to melt the ice accumulated in overnight snow storms.

And signals your coffee maker to turn on 5 minutes late as well.

Source: https://blogs.cisco.com/diversity/the-internet-of-things-infographic

Knowledge Base Construction



Scalable Probabilistic Databases with Factor Graphs and MCMC

Michael Wick
University of Massachusetts
Computer Science
140 Governor's Drive
Amherst, MA
mwick@cs.umass.edu

Andrew McCallum
University of Massachusetts
Computer Science
140 Governor's Drive
Amherst, MA
mccallum@cs.umass.edu

Gerome Miklau
University of Massachusetts
Computer Science
140 Governor's Drive
Amherst, MA
miklau@cs.umass.edu

ABSTRACT

Incorporating probabilities into the semantics of incomplete databases has posed many challenges, forcing systems to sacrifice modeling power, scalability, or treatment of relational algebra operators. We propose an alternative approach where the underlying relational database always represents a single world, and an external factor graph encodes a distribution over possible worlds; Markov chain Monte Carlo (MCMC) inference is then used to recover this uncertainty to a desired level of fidelity. Our approach allows the efficient evaluation of arbitrary queries over probabilistic databases with arbitrary dependencies expressed by graphical models with structure that changes during inference. MCMC sampling provides efficiency by hypothesizing *modifications* to possible worlds rather than generating entire worlds from scratch. Queries are then run over the portions of the world that change, avoiding the onerous cost of running full queries over each sampled world. A significant innovation of this work is the connection between MCMC sampling and materialized view maintenance techniques: we find empirically that using view maintenance techniques is several orders of magnitude faster than naively querying each sampled world. We also demonstrate our system's ability to answer relational queries with aggregation, and demonstrate additional scalability through the use of parallelization on a real-world complex model of information extraction. This framework is sufficiently expressive to support probabilistic inference not only for answering queries, but also for inferring missing database content from raw evidence.

current PDBs do not achieve the difficult balance of expressivity and efficiency necessary to support such a range of scalable real-world structured prediction systems.

Indeed, there is an inherent tension between the expressiveness of a representation system and the efficiency of query evaluation. Many recent approaches to probabilistic databases can be characterized as residing on either pole of this continuum. For example, some systems favor efficient query evaluation by restricting modeling power with strict independence assumptions [5, 6, 1]. Other systems allow rich representations that render query evaluation intractable for a large portion of their model family [10, 24, 19, 20].
In this paper we
to provide a powe
query evaluation

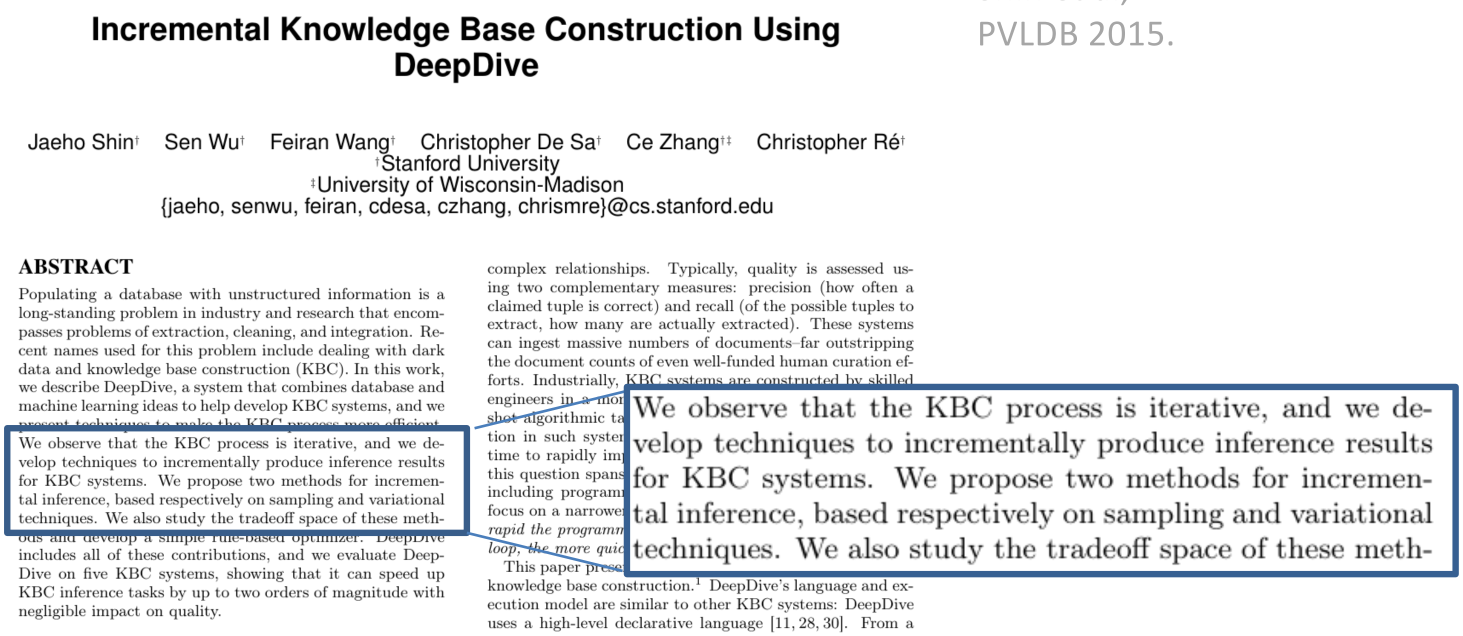
Graphical mod
uncertainty and p
plications, includ
language process
traction [16], and
ing even more ac
purpose probabili
graphs are a parti
that serve as an u
and Markov random
ponential family probability distribution.

In our approach, we use factor graphs to represent uncertainty over our relational data, and MCMC for inference of database con-

over the portions of the world that change, avoiding the onerous cost of running full queries over each sampled world. A significant innovation of this work is the connection between MCMC sampling and materialized view maintenance techniques: we find empirically that using view maintenance techniques is several orders of magnitude faster than naively querying each sampled world. We

Wick, McCallum, Miklau, PVLDB 2010.

Knowledge Base Construction



Incremental Knowledge Base Construction Using DeepDive

Jaeho Shin[†] Sen Wu[†] Feiran Wang[†] Christopher De Sa[†] Ce Zhang[†‡] Christopher Ré[†]
[†]Stanford University
[‡]University of Wisconsin-Madison
{jaeho, senwu, feiran, cdesa, czhang, chrismre}@cs.stanford.edu

ABSTRACT

Populating a database with unstructured information is a long-standing problem in industry and research that encompasses problems of extraction, cleaning, and integration. Recent names used for this problem include dealing with dark data and knowledge base construction (KBC). In this work, we describe DeepDive, a system that combines database and machine learning ideas to help develop KBC systems, and we present techniques to make the KBC process more efficient. We observe that the KBC process is iterative, and we develop techniques to incrementally produce inference results for KBC systems. We propose two methods for incremental inference, based respectively on sampling and variational techniques. We also study the tradeoff space of these methods and develop a simple rule-based optimizer. DeepDive includes all of these contributions, and we evaluate DeepDive on five KBC systems, showing that it can speed up KBC inference tasks by up to two orders of magnitude with negligible impact on quality.

complex relationships. Typically, quality is assessed using two complementary measures: precision (how often a claimed tuple is correct) and recall (of the possible tuples to extract, how many are actually extracted). These systems can ingest massive numbers of documents–far outstripping the document counts of even well-funded human curation efforts. Industrially, KBC systems are constructed by skilled engineers in a mor
shot algorithmic ta
tion in such syste
time to rapidly im
this question spans
including programm
focus on a narrowe
rapid the programm
loop, the more quic

This paper prese

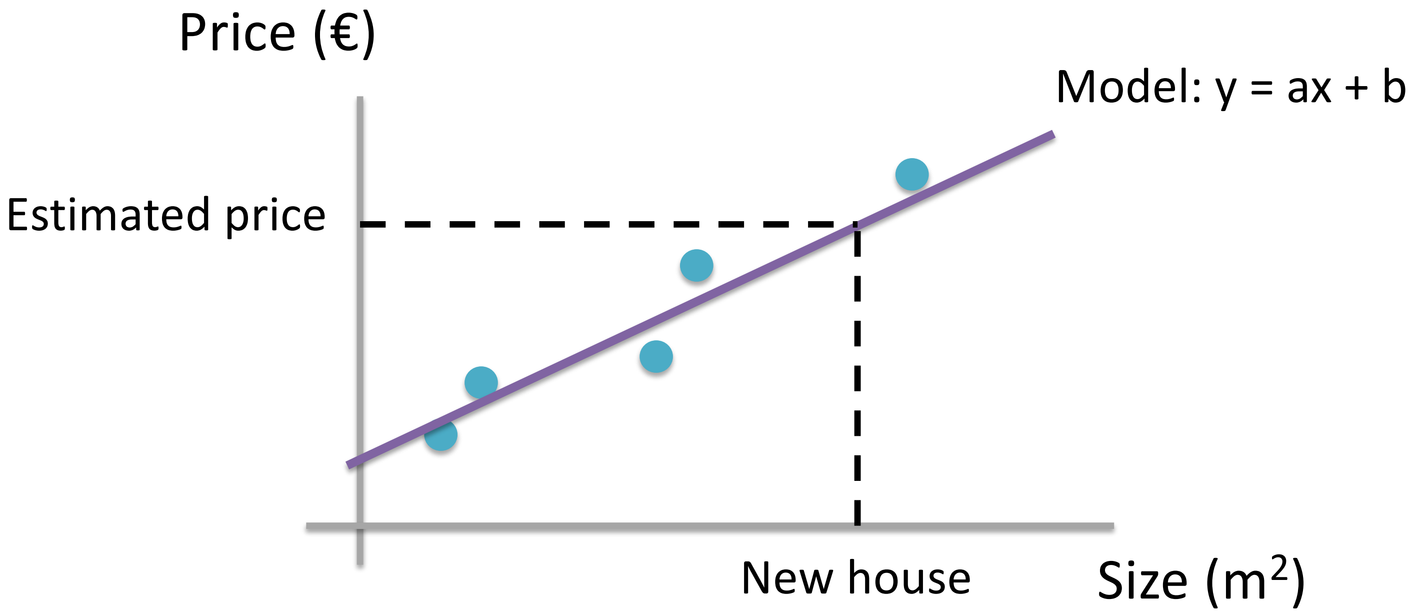
knowledge base construction.[1] DeepDive's language and execution model are similar to other KBC systems: DeepDive uses a high-level declarative language [11, 28, 30]. From a

We observe that the KBC process is iterative, and we develop techniques to incrementally produce inference results for KBC systems. We propose two methods for incremental inference, based respectively on sampling and variational techniques. We also study the tradeoff space of these meth-

Shin et al,
PVLDB 2015.

Online Machine Learning

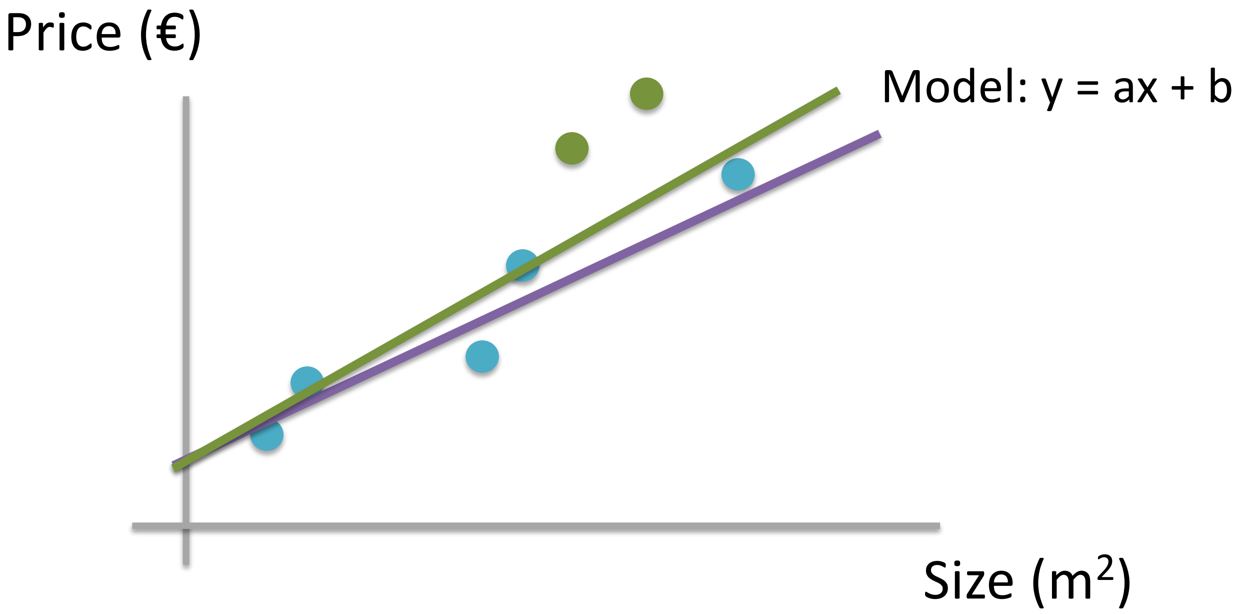
Estimating House Prices



Linear regression with parameters *(a,b)*

Online Machine Learning

Estimating House Prices



Linear regression with parameters *(a,b)*

Conclusion:

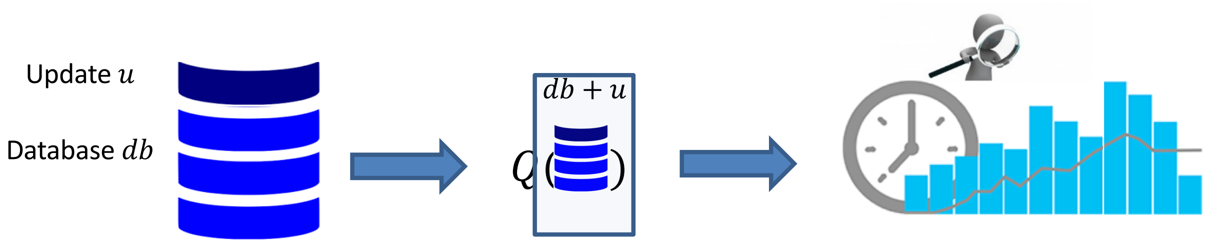
Dynamic Query Processing is pervasive in a wide range of application areas.

Outline

- Part I: Introduction
- Part II: Main Algorithmic Ideas in Dynamic Query Processing: Traditional IVM and Recent Advances
- Part III: Generalizations to Arbitrary Ring Structures
- Part IV: Dynamic Query Processing in Big Data Frameworks
- Part V: Outlook

Outline

- Part I: Introduction
- **Part II: Main Algorithmic Ideas in Dynamic Query Processing: Traditional IVM and Recent Advances**
- Part III: Generalizations to Arbitrary Ring Structures
- Part IV: Dynamic Query Processing in Big Data Frameworks
- Part V: Outlook



Update u

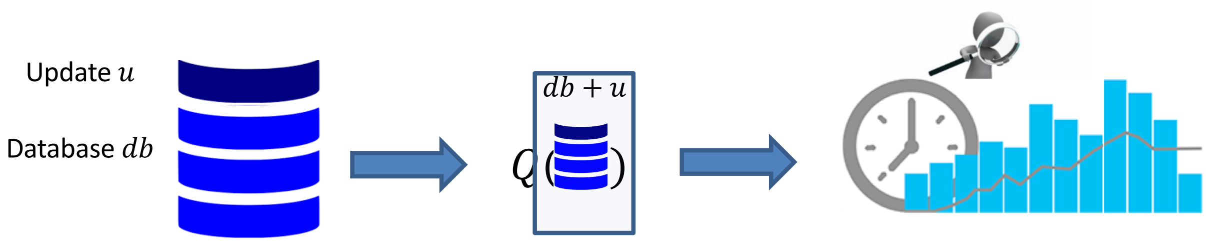
Database db

$db + u$

$Q($ $)$

Dynamic query evaluation

Avoid full recomputation – compute incrementally



Update u

Database db

$Q(db + u)$

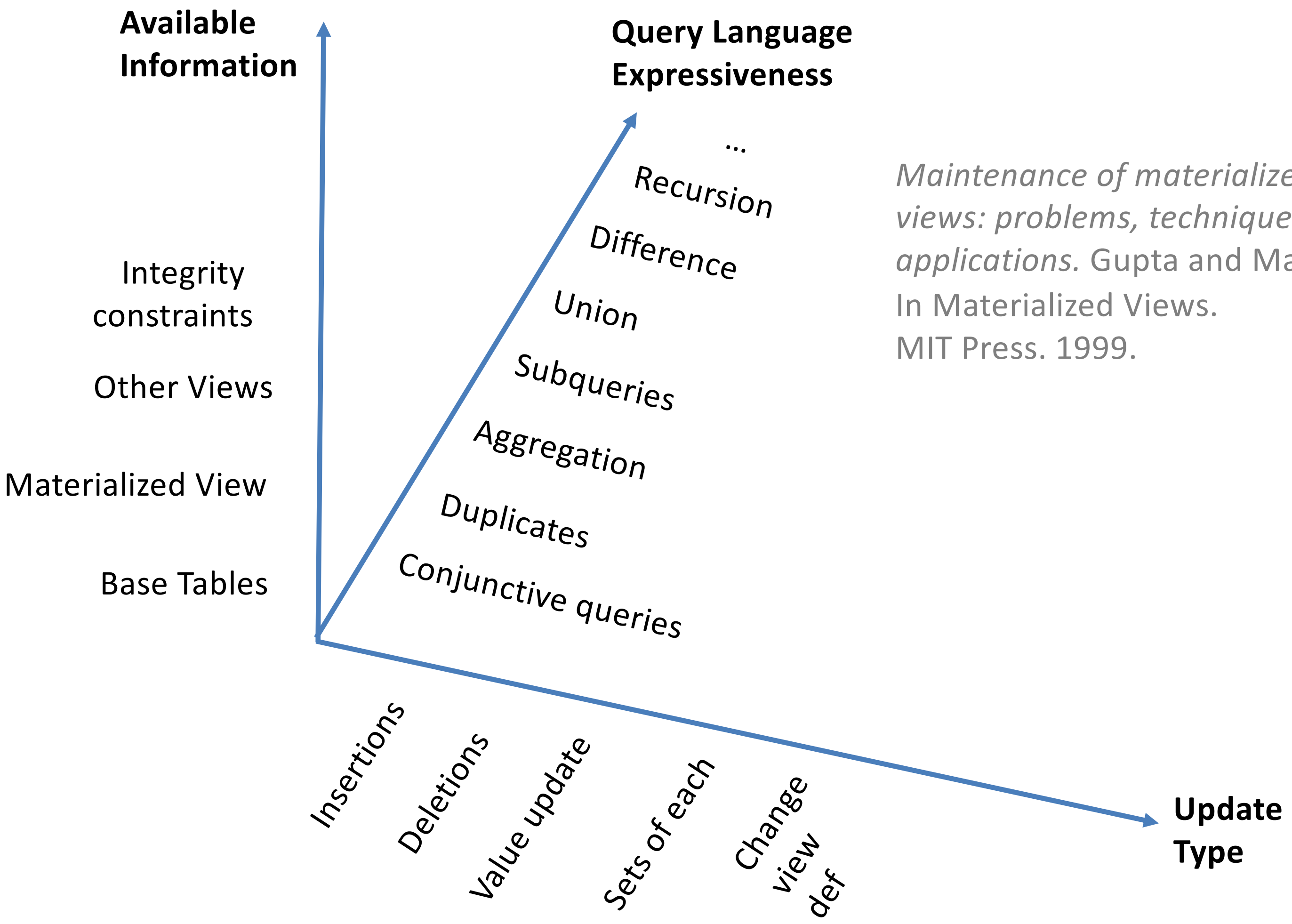
Base data

Query

View = Cache

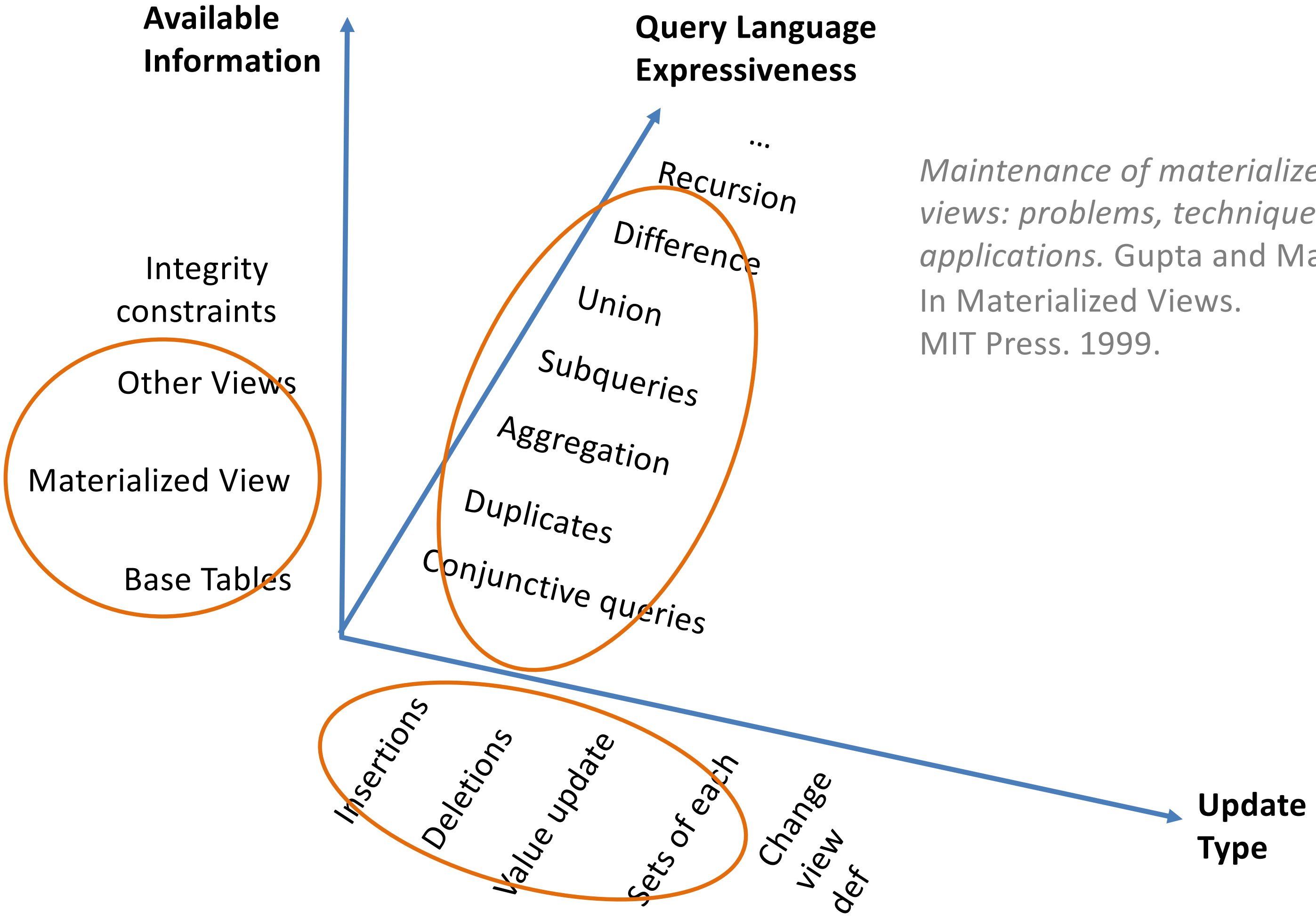
Incremental View Maintenance (IVM)

Dimensions of IVM



Maintenance of materialized views: problems, techniques, applications. Gupta and Mannick.
In Materialized Views.
MIT Press. 1999.

Dimensions of IVM: this tutorial



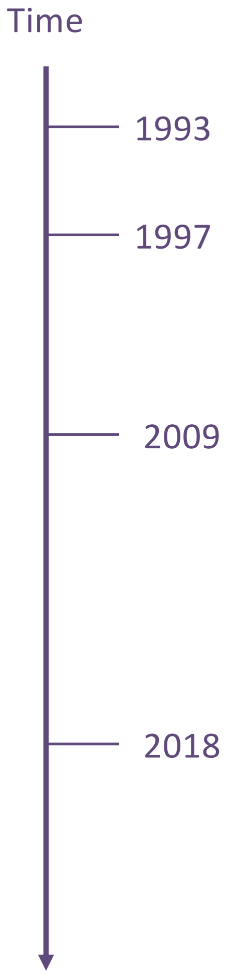
Maintenance of materialized views: problems, techniques, applications. Gupta and Mannick.
In Materialized Views.
MIT Press. 1999.

Main Algorithmic Ideas

1. IVM ≡ processing of delta queries

2. Materialize results of subqueries in addition to the actual query result

3. Exploit data skew



Time

1993

1997

2009

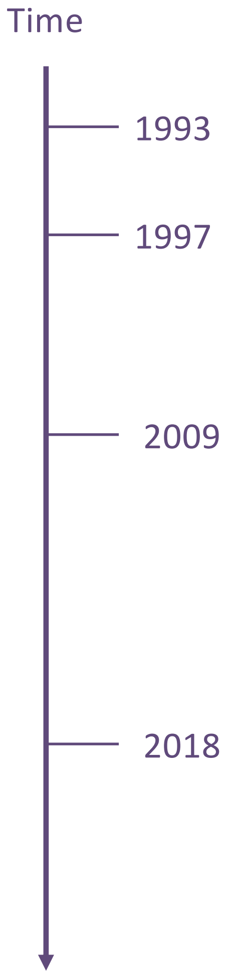
2018

Main Algorithmic Ideas

1. IVM ≡ processing of delta queries

2. Materialize results of subqueries in addition to the actual query result

3. Exploit data skew



Time

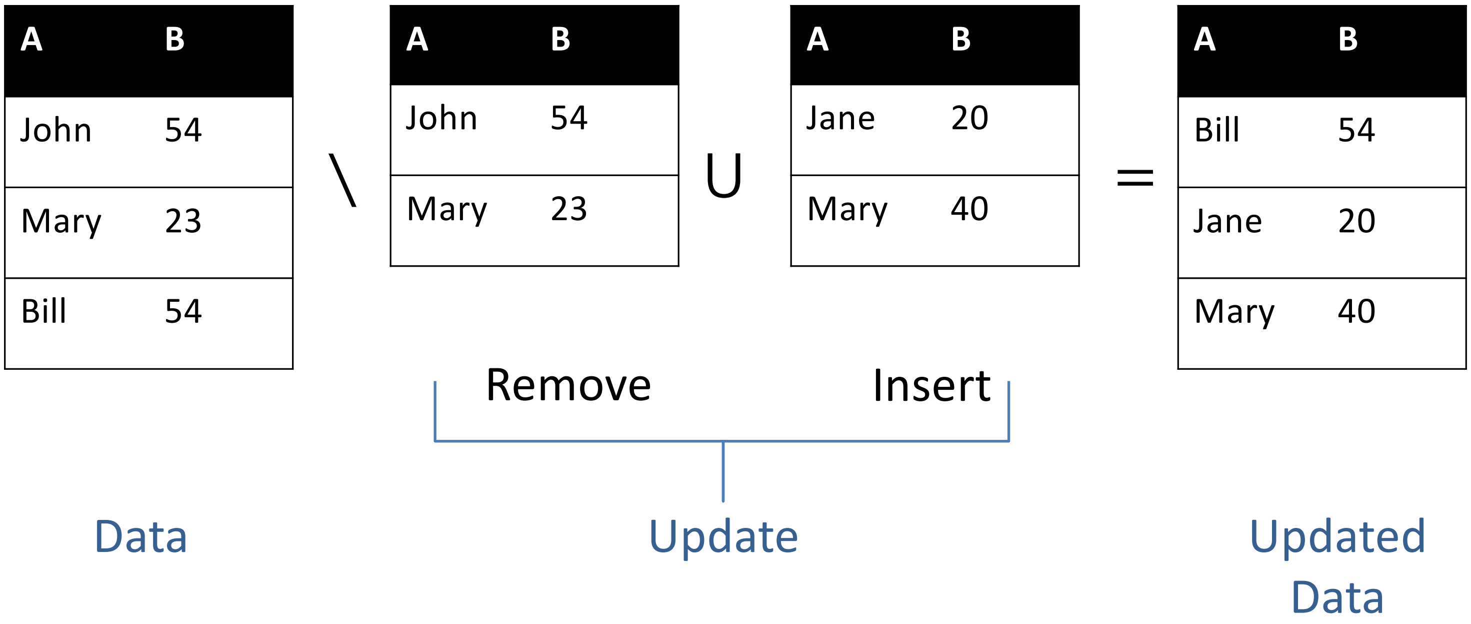
1993

1997

2009

2018

Traditional update representation



A	B
John	54
Mary	23
Bill	54

\

A	B
John	54
Mary	23

U

A	B
Jane	20
Mary	40

=

A	B
Bill	54
Jane	20
Mary	40

Remove

Insert

Data

Update

Updated Data

Uniform update representation

R

A	B	CNT
John	54	2
Mary	23	1
Bill	54	4

Data

$+$

ΔR

A	B	CNT
John	54	-1
Mary	23	-1
Jane	20	2
Mary	40	1

Update

$=$

$R + \Delta R$

A	B	CNT
John	54	1
Bill	54	4
Jane	20	2
Mary	40	1

Updated Data

Negative = remove
Multiplicity
Positive = add
(Multiplicity 0 = absent)

Query semantics (1/5)

Selection

$\sigma_{B>30}$

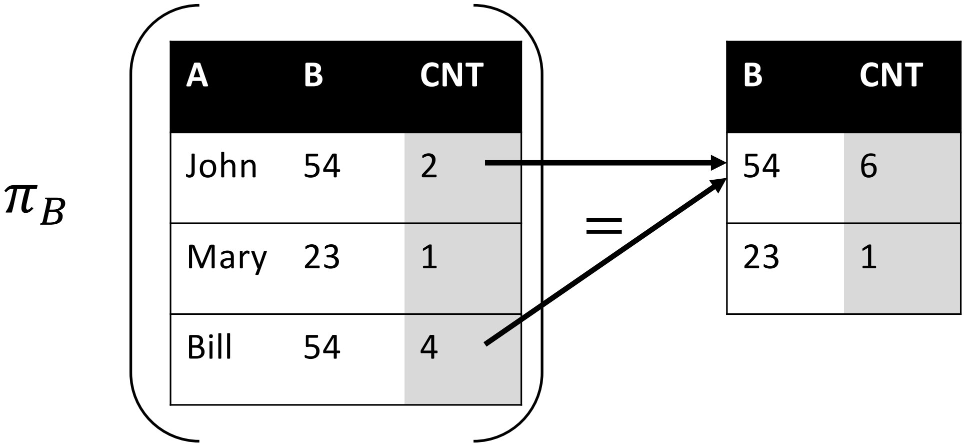
A	B	CNT
John	54	2
Mary	23	1
Bill	54	4

$=$

A	B	CNT
John	54	2
Bill	54	4

Query semantics (2/5)

Projection



π_B

A	B	CNT
John	54	2
Mary	23	1
Bill	54	4

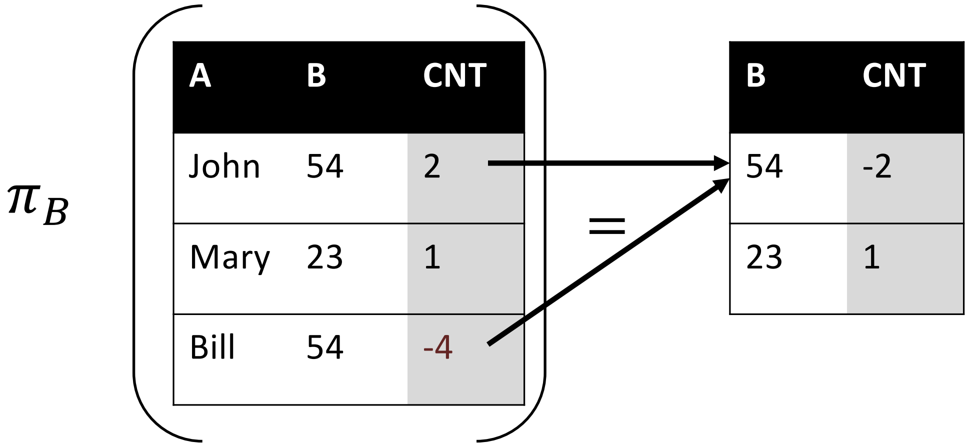
$=$

B	CNT
54	6
23	1

Duplicate-preserving bag-based projection

Query semantics (2/5)

Projection



π_B

A	B	CNT
John	54	2
Mary	23	1
Bill	54	-4

$=$

B	CNT
54	-2
23	1

Duplicate-preserving bag-based projection

Query semantics (3/5)

Union

A	B	CNT
John	54	2
Mary	23	1
Bill	54	4

+

A	B	CNT
John	54	-1
Mary	23	-1
Jane	20	2
Mary	40	1

=

A	B	CNT
John	54	1
Bill	54	4
Jane	20	2
Mary	40	1

Duplicate-preserving bag-based union

Query semantics (4/5)

Difference

A	B	CNT
John	54	2
Mary	23	1
Bill	54	4

−

A	B	CNT
John	54	-1
Mary	23	-1
Jane	20	2
Mary	40	1

=

A	B	CNT
John	54	3
Mary	23	2
Bill	54	4
Jane	20	-2
Mary	40	-1

This is *not* bag difference!

Query semantics (5/5)

Join

× =

A	B	CNT
John	54	2
Mary	23	1
Bill	54	4

⋈

B	C	CNT
54	Gold	10
20	Silver	5

=

A	B	C	CNT
John	54	Gold	20
Bill	54	Silver	40

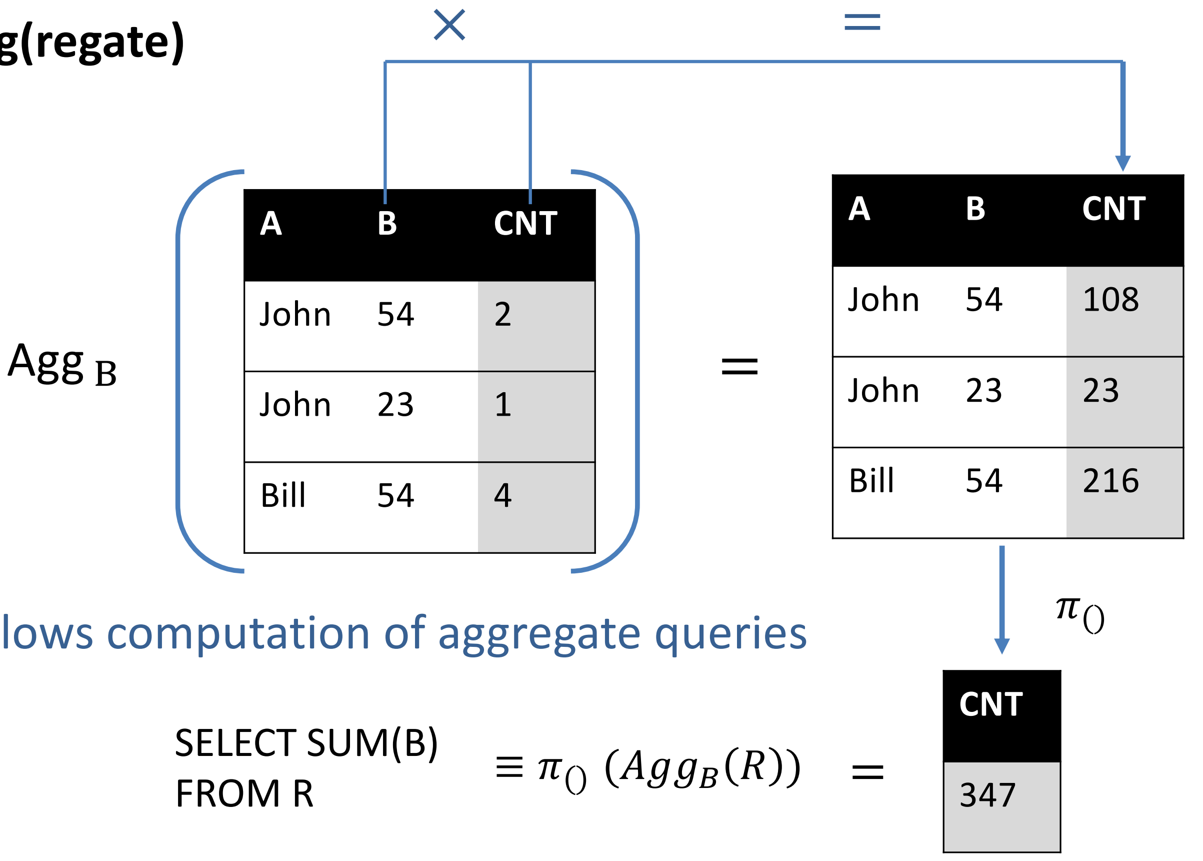
Multiply multiplicities of joining tuples

Observations

- Query evaluation algorithms are trivially modified to compute the CNT values.
- Under this modified semantics, each tuple in the query result specifies the number of derivations for that tuple.

Query semantics: aggregation

Agg(regate)



Agg_{B}

A	B	CNT
John	54	2
John	23	1
Bill	54	4

=

A	B	CNT
John	54	108
John	23	23
Bill	54	216

Allows computation of aggregate queries

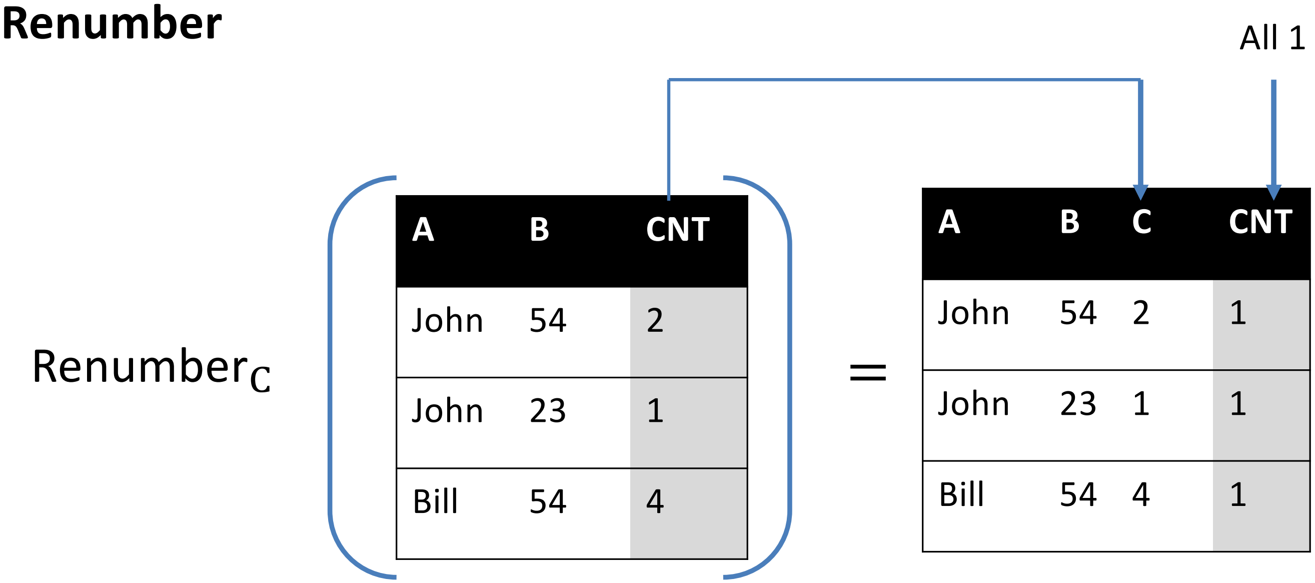
SELECT SUM(B)
FROM R

$\equiv \pi_{()}\ (Agg_B(R))$ =

CNT
347

Query semantics: aggregation

Renumber



Renumber$_C$

A	B	CNT
John	54	2
John	23	1
Bill	54	4

=

A	B	C	CNT
John	54	2	1
John	23	1	1
Bill	54	4	1

Allows computation of groupby + aggregate queries

Groupby: Example

SELECT SUM(B)
FROM R
GROUP BY A

A	B	CNT
John	54	2
John	23	1
Bill	54	4

→ Agg $_{\mathrm{B}}$

A	B	CNT
John	54	108
John	23	23
Bill	54	216

→ π_A

A	CNT
John	131
Bill	216

→ Renumber $_{\mathrm{SUM(B)}}$

A	SUM (B)	CNT
John	131	1
Bill	216	1

Delta queries

Multiplicity union

$$\begin{tikzcd}
\boldsymbol{D} \arrow[r, "\text{Update } \Delta\boldsymbol{D}"] \arrow[d, "\text{Query } Q"'] & \boldsymbol{D} + \Delta\boldsymbol{D} \arrow[d, "\text{Recompute } Q"] \\
Q(\boldsymbol{D}) \arrow[r, "\text{Incremental maintenance } \Delta T"'] & Q(\boldsymbol{D} + \Delta\boldsymbol{D}) = Q(\boldsymbol{D}) \boxed{+ \Delta T}
\end{tikzcd}$$

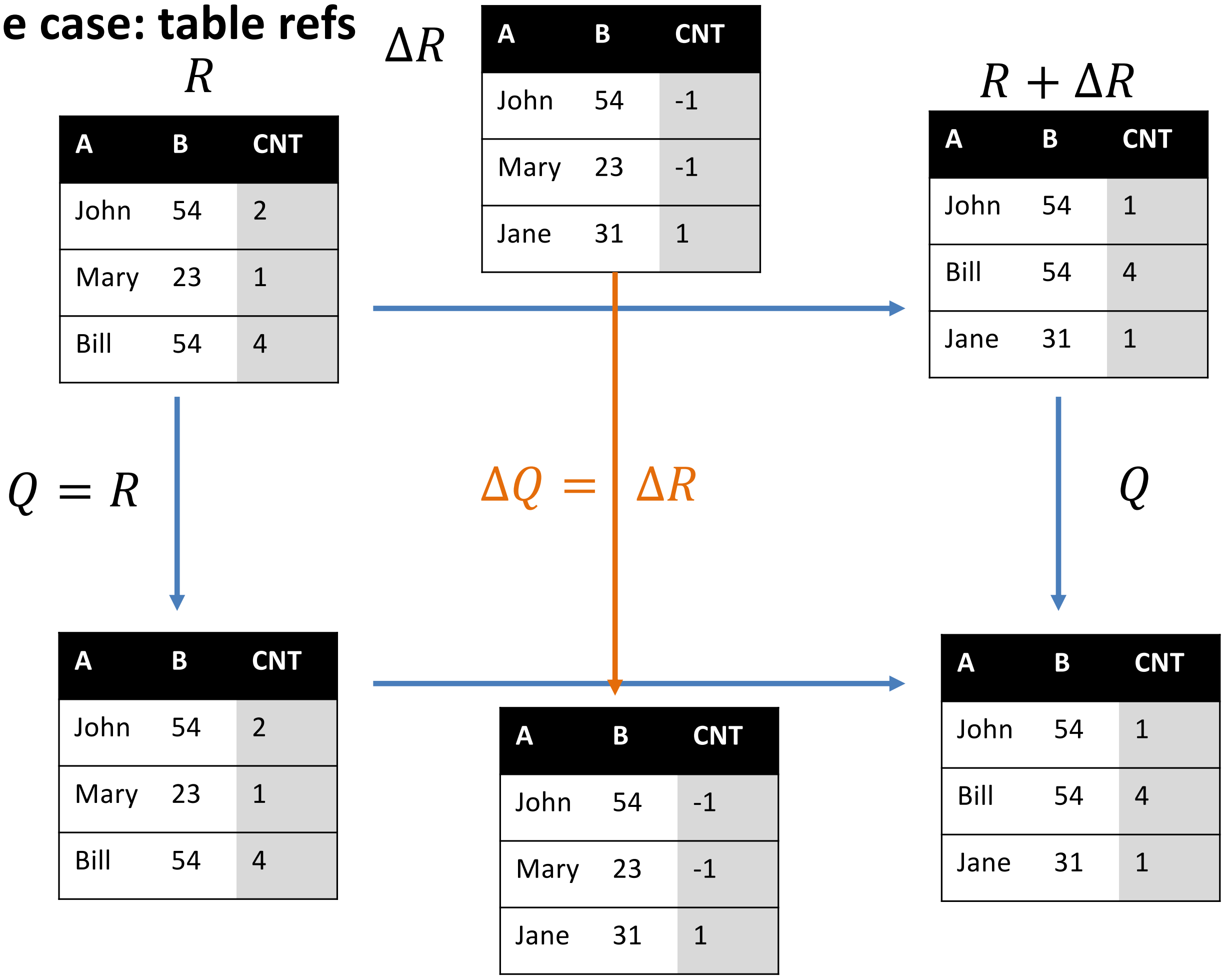
Key insight:

For every query Q in relational algebra (with multiplicities), it is possible to write a query ΔQ that operates on the old database $\boldsymbol{D}$ and the update $\Delta\boldsymbol{D}$ s.t.

$$Q(\boldsymbol{D} + \Delta\boldsymbol{D}) = Q(\boldsymbol{D}) + \boxed{\Delta Q(\boldsymbol{D}, \Delta\boldsymbol{D})}$$

Delta queries

Base case: table refs



R

A	B	CNT
John	54	2
Mary	23	1
Bill	54	4

ΔR

A	B	CNT
John	54	-1
Mary	23	-1
Jane	31	1

$R + \Delta R$

A	B	CNT
John	54	1
Bill	54	4
Jane	31	1

$Q = R$

$\Delta Q = \Delta R$

Q

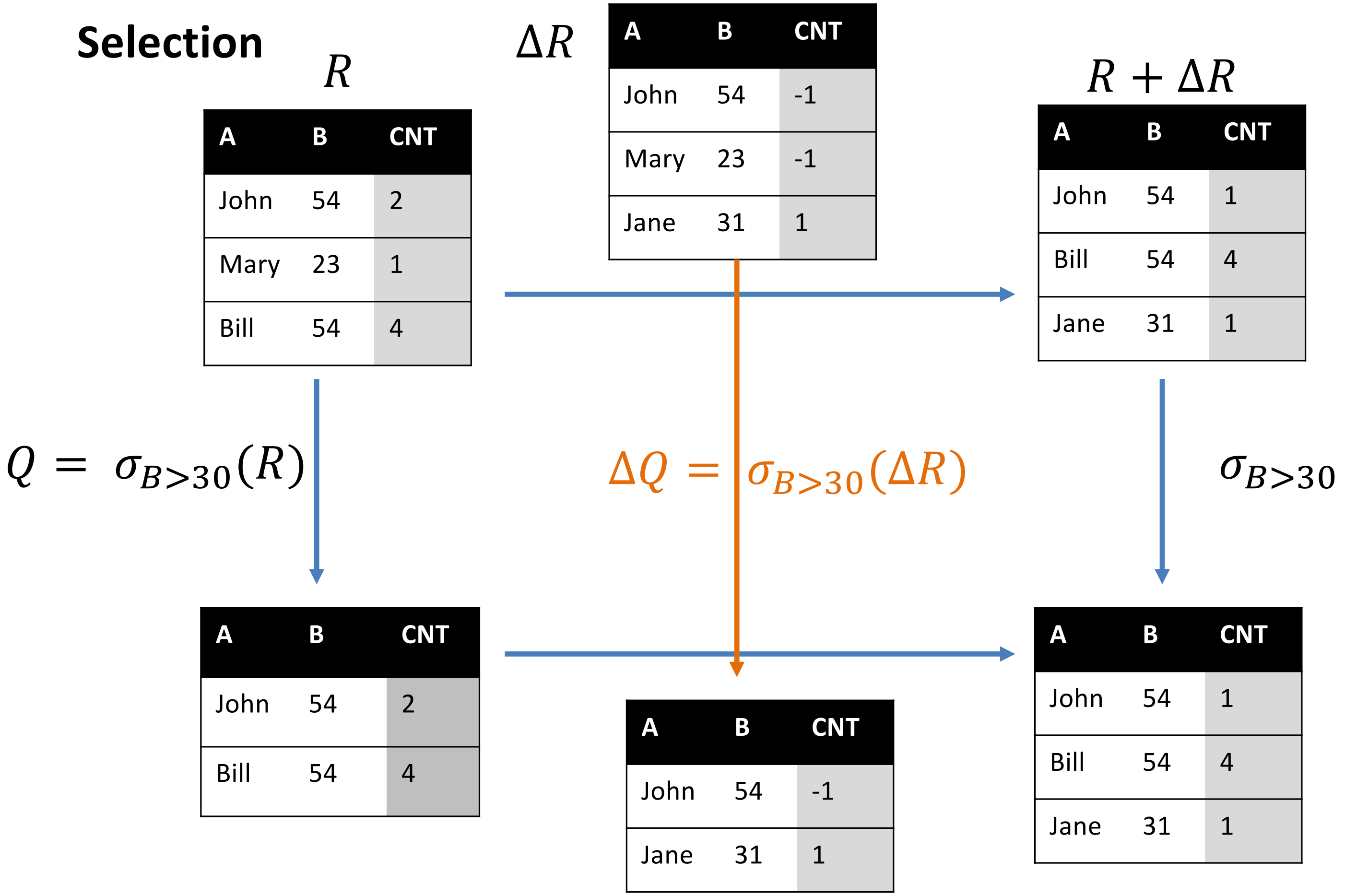
A	B	CNT
John	54	2
Mary	23	1
Bill	54	4

A	B	CNT
John	54	-1
Mary	23	-1
Jane	31	1

A	B	CNT
John	54	1
Bill	54	4
Jane	31	1

Delta queries

Selection



R

A	B	CNT
John	54	2
Mary	23	1
Bill	54	4

ΔR

A	B	CNT
John	54	-1
Mary	23	-1
Jane	31	1

$R + \Delta R$

A	B	CNT
John	54	1
Bill	54	4
Jane	31	1

$Q = \sigma_{B>30}(R)$

$\Delta Q = \sigma_{B>30}(\Delta R)$

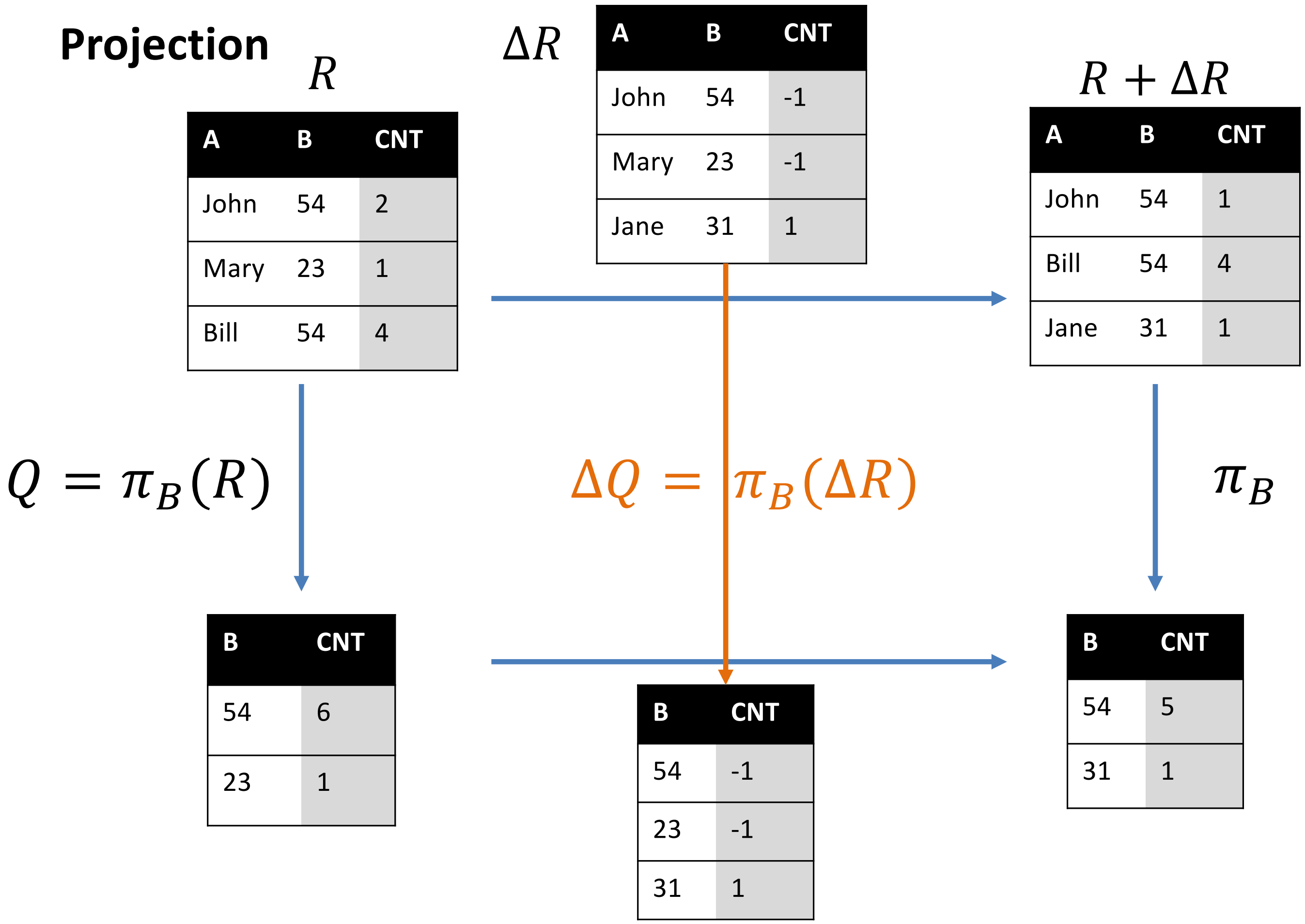
$\sigma_{B>30}$

A	B	CNT
John	54	2
Bill	54	4

A	B	CNT
John	54	-1
Jane	31	1

A	B	CNT
John	54	1
Bill	54	4
Jane	31	1

Delta queries



Projection

R

A	B	CNT
John	54	2
Mary	23	1
Bill	54	4

ΔR

A	B	CNT
John	54	-1
Mary	23	-1
Jane	31	1

$R + \Delta R$

A	B	CNT
John	54	1
Bill	54	4
Jane	31	1

$Q = \pi_B(R)$

$\Delta Q = \pi_B(\Delta R)$

π_B

B	CNT
54	6
23	1

B	CNT
54	-1
23	-1
31	1

B	CNT
54	5
31	1

Delta queries

Join $Q = R \bowtie S$

R

A	B	CNT
John	54	2
Mary	23	1
Bill	54	4

$+$

ΔR

A	B	CNT
John	54	-2
Jane	32	1

$\bowtie$

S

B	C	CNT
54	Gold	10
20	Silver	5

$+$

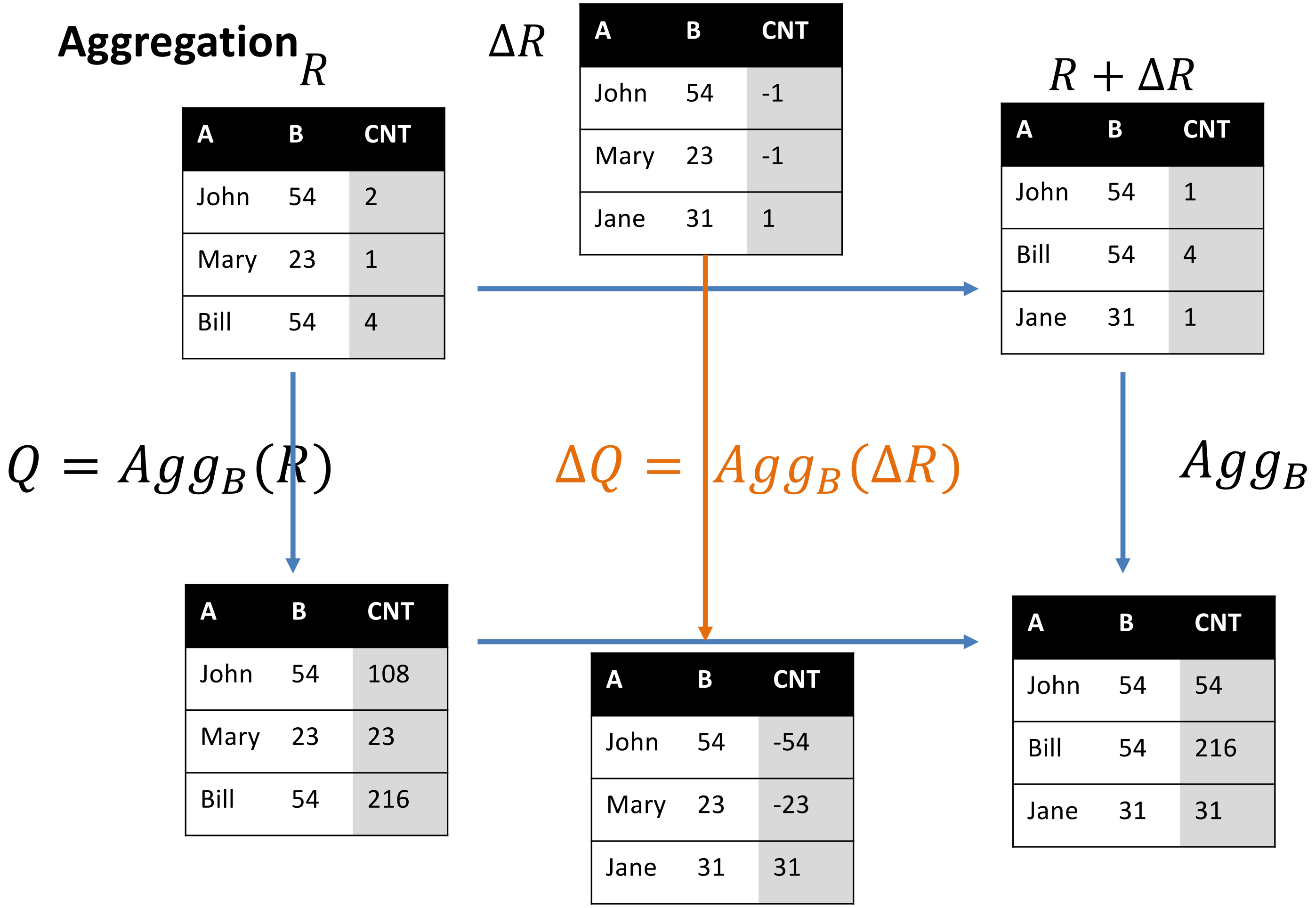
ΔS

B	C	CNT
54	Gold	-2
23	Silver	2

$Q(\boldsymbol{D} + \Delta\boldsymbol{D}) = (R + \Delta R) \bowtie (S + \Delta S)$ **Distributivity of + over ⋈**

$$= \underbrace{\boxed{(R \bowtie S)}}_{Q(\boldsymbol{D})} + \underbrace{\boxed{(\Delta R \bowtie S) + (R \bowtie \Delta S) + (\Delta R \bowtie \Delta S)}}_{\Delta Q(\boldsymbol{D}, \Delta\boldsymbol{D})}$$

Delta queries



Aggregation R

A	B	CNT
John	54	2
Mary	23	1
Bill	54	4

ΔR

A	B	CNT
John	54	-1
Mary	23	-1
Jane	31	1

$R + \Delta R$

A	B	CNT
John	54	1
Bill	54	4
Jane	31	1

$Q = Agg_B(R)$

$\Delta Q = Agg_B(\Delta R)$

Agg_B

A	B	CNT
John	54	108
Mary	23	23
Bill	54	216

A	B	CNT
John	54	-54
Mary	23	-23
Jane	31	31

A	B	CNT
John	54	54
Bill	54	216
Jane	31	31

Delta queries

Renumber $Q = Renumber_C(R)$

$Q\Bigg($

R

A	B	CNT
John	54	2
John	23	1
Bill	54	4

$+$

ΔR

A	B	CNT
John	54	-2
Jane	32	1

$\Bigg)$

$=$

A	B	C	CNT
John	23	1	1
Bill	54	4	1
Jane	32	1	1

$$\Delta \text{Renumber}_C(R, \Delta R)$$

$$= \text{Renumber}_C\ (R + \Delta R) \text{ - } \text{Renumber}_C(R)$$

Recomputation necessary!
Specialized algorithm possible

Delta queries: summary

Query Q(D)	Delta Query $\Delta \mathrm{Q}(\mathrm{D}, \Delta \mathrm{D})$
Table R	Table ΔR
$\sigma_\theta(Q')$	$\sigma_\theta(\Delta Q')$
$\pi_{\vec{A}}(Q')$	$\pi_{\vec{A}}(\Delta Q')$
$Q_1 + Q_2$	$\Delta Q_1 + \Delta Q_2$
$Q_1 \bowtie Q_2$	$\Delta Q_1 \bowtie Q_2 + Q_1 \bowtie \Delta Q_2 + \Delta Q_1 \bowtie \Delta Q_2$
$\mathrm{Agg}_\mathrm{A}(Q')$	$\mathrm{Agg}_\mathrm{A}(\Delta Q')$
$\mathrm{Renumber}_\mathrm{A}$(Q’)	$\mathrm{Renumber}_\mathrm{A}$(Q’ + $\Delta Q'$) - $\mathrm{Renumber}_\mathrm{A}$(Q’)

The Counting Algorithm

Maintaining Views Incrementally.
Gupta, Mumick, Subrahmaniam. SIGMOD 1993.

- Store all relations in database $\boldsymbol{D}$
- Store (materialize) $Q(\boldsymbol{D})$ in view V
- Upon update $\Delta\boldsymbol{D}$:
  - Use ΔQ to compute $\Delta Q(\boldsymbol{D}, \Delta\boldsymbol{D})$
  - Add $\Delta Q(\boldsymbol{D}, \Delta\boldsymbol{D})$ to V

Use your favorite Query Evaluation algorithm to evaluate this. ΔD is expected to be small!

The Counting Algorithm: an example

$$Q = \pi_{AD}(R \bowtie S \bowtie T)$$

R

A	B	CNT
John	54	2
Mary	23	1
Bill	54	4

ΔR

A	B	CNT
John	54	-2
Mary	23	2

S

B	C	CNT
54	Gold	2
20	Silver	1

T

C	D	CNT
Gold	100	2
Gold	80	1
Silver	50	1
Bronze	20	4

$Q(\boldsymbol{D})$

A	D	CNT
John	100	8
John	80	4
Bill	100	16
Bill	80	8

$\Delta Q(\boldsymbol{D}, \Delta\boldsymbol{D})$

A	D	CNT
John	100	-8
John	80	-4

$$\Delta Q = \pi_{AD}(\Delta R \bowtie S \bowtie T \left.\begin{aligned} &+(R + \Delta R) \bowtie \Delta S \bowtie T \\ &+(R + \Delta R) \bowtie (S + \Delta S) \bowtie \Delta T) \end{aligned}\right\} \text{Empty!}$$

The Counting Algorithm: an example

$$Q = \pi_{AD}(R \bowtie S \bowtie T)$$

R

A	B	CNT
John	54	2
Mary	23	1
Bill	54	4

ΔR

A	B	CNT
John	54	-2
Mary	23	2

S

B	C	CNT
54	Gold	2
20	Silver	1

T

C	D	CNT
Gold	100	2
Gold	80	1
Silver	50	1
Bronze	20	4

$Q(D)$

A	D	CNT
John	100	8
John	80	4
Bill	100	16
Bill	80	8

$\Delta Q(\boldsymbol{D}, \Delta \boldsymbol{D})$

A	D	CNT
John	100	-8
John	80	-4

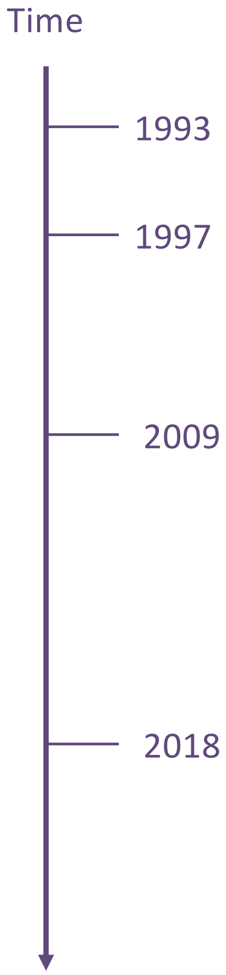
$$\Delta_R Q = \pi_{AD}(\Delta R \bowtie S \bowtie T)$$

Main Algorithmic Ideas

1. IVM ≡ processing of delta queries

2. Materialize results of subqueries in addition to the actual query result

3. Exploit data skew



Time

1993

1997

2009

2018

Delta evaluation through recomputation?

$Q = \pi_{AD}(R \bowtie S \bowtie T)$

Maintain under updates to R

R

A	B	CNT
John	54	2
Mary	23	1
Bill	54	4

S

B	C	CNT
54	Gold	2
20	Silver	1

T

C	D	CNT
Gold	100	2
Gold	80	1
Silver	50	1
Bronze	20	4

$Q(\boldsymbol{D})$

A	D	CNT
John	100	8
John	80	4
Bill	100	16
Bill	80	8

ΔR

A	B	CNT
John	54	-2
Mary	23	2

$\Delta Q(\boldsymbol{D}, \Delta \boldsymbol{D})$

A	D	CNT
John	100	-8
John	80	-4

$\Delta_R Q = \pi_{AD}(\Delta R \bowtie S \bowtie T)$

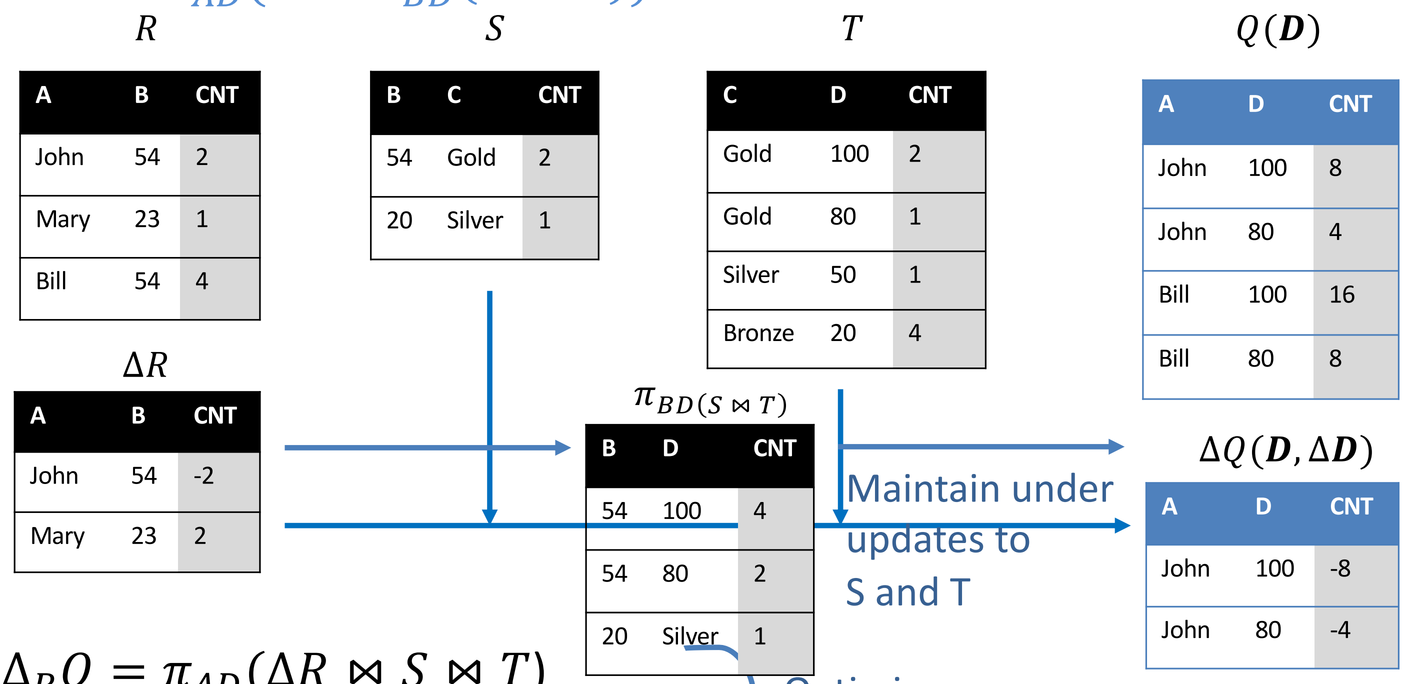
- **The join $S \bowtie T$ is recomputed for every update ΔR**
- **Update latency may be high**

Key Insight

$$Q = \pi_{AD}(R \bowtie S \bowtie T)$$

$$\equiv \pi_{AD}(R \bowtie \pi_{BD}(S \bowtie T))$$

Maintain under updates to R



R

A	B	CNT
John	54	2
Mary	23	1
Bill	54	4

S

B	C	CNT
54	Gold	2
20	Silver	1

T

C	D	CNT
Gold	100	2
Gold	80	1
Silver	50	1
Bronze	20	4

$Q(\mathbf{D})$

A	D	CNT
John	100	8
John	80	4
Bill	100	16
Bill	80	8

ΔR

A	B	CNT
John	54	-2
Mary	23	2

$\pi_{BD(S \bowtie T)}$

B	D	CNT
54	100	4
54	80	2
20	Silver	1

$\Delta Q(\mathbf{D}, \Delta \mathbf{D})$

A	D	CNT
John	100	-8
John	80	-4

Maintain under updates to S and T

$$\Delta_R Q = \pi_{AD}(\Delta R \bowtie S \bowtie T)$$

$$\equiv \pi_{AD}(\Delta R \bowtie \underset{Q_{ST}}{\boxed{\pi_{BD}(S \bowtie T)}})$$

Optimize

Materialize as auxiliary view to avoid recomputation

Key Insight

$$Q_{ST} = \pi_{BD}(S \bowtie T)$$

How to maintain Q_{ST}

R

A	B	CNT
John	54	2
Mary	23	1
Bill	54	4

S

B	C	CNT
54	Gold	2
20	Silver	1

T

C	D	CNT
Gold	100	2
Gold	80	1
Silver	50	1
Bronze	20	4

$\pi_{BD(S \bowtie T)}$

B	D	CNT
54	100	4
54	80	2
20	Silver	1

$$\Delta_{\mathrm{S}} Q_{ST} = \pi_{BD}(\Delta S \bowtie \boxed{T})$$

$$\Delta_{\mathrm{T}} Q_{ST} = \pi_{BD}(\boxed{S} \bowtie \Delta T)$$

No auxiliary view necessary (base table)
Trivial to maintain under updates

(When really a subquery: continue reasoning)

What are we doing here ?

$Q = \pi_{AD}(R \bowtie S \bowtie T)$ Maintain under updates to R

$\Delta_R Q = \pi_{AD}(\Delta R \bowtie \boxed{\pi_{BD}(S \bowtie T)})$ Materialize as aux. view

$\boxed{Q_{ST} = \pi_{BD}(S \bowtie T)}$ Maintain under updates to S, T

$\Delta_S Q_{ST} = \pi_{BD}(\Delta S \bowtie \boxed{T})$ Materialize T

$\Delta_T Q_{ST} = \pi_{BD}(\boxed{S} \bowtie \Delta T)$ Materialize S

First-order Delta Query

What are we doing here ?

$Q = \pi_{AD}(R \bowtie S \bowtie T)$ Maintain under updates to R

$\Delta_R Q = \pi_{AD}(\Delta R \bowtie \boxed{\pi_{BD}(S \bowtie T)})$ Materialize as aux. view

$\boxed{Q_{ST} = \pi_{BD}(S \bowtie T)}$ Maintain under updates to S, T

$$\left.\begin{aligned} \Delta_S Q_{ST} &= \pi_{BD}(\Delta S \bowtie \boxed{T}) \\ \Delta_T Q_{ST} &= \pi_{BD}(\boxed{S} \bowtie \Delta T) \end{aligned}\right\}$$

Materialize T

Materialize S

Delta of (subquery of)
a Delta
≡
Higher-Order Delta

Continuing our reasoning

$$Q = \pi_{AD}(R \bowtie S \bowtie T)$$

$$\equiv \pi_{AD}(\pi_{AC}(R \bowtie S) \bowtie T)$$

Maintain under updates to T

R

A	B	CNT
John	54	2
Mary	23	1
Bill	54	4

S

B	C	CNT
54	Gold	2
20	Silver	1

T

C	D	CNT
Gold	100	2
Gold	80	1
Silver	50	1
Bronze	20	4

$Q(\boldsymbol{D})$

A	D	CNT
John	100	8
John	80	4
Bill	100	16
Bill	80	8

$\pi_{AC(R \bowtie S)}$

A	C	CNT
John	Gold	4
Bill	Gold	8

Maintain under updates to R and S

ΔT

C	D	CNT
Gold	100	-1
Bronze	20	2

$\Delta_T Q(\boldsymbol{D}, \Delta\boldsymbol{D})$

A	D	CNT
John	100	-4
John	100	-8

$$\Delta_T Q = \pi_{AD}(R \bowtie S \bowtie \Delta T)$$

$$\equiv \pi_{AD}(\boxed{\pi_{AC}(R \bowtie S)} \bowtie \Delta T))$$

Materialize as auxiliary view to avoid recomputation

Continuing our reasoning

$$Q = \pi_{AD}(R \bowtie S \bowtie T)$$

$$\equiv \pi_{AD}(R \bowtie T \bowtie S)$$

Maintain under updates to S

R

A	B	CNT
John	54	2
Mary	23	1
Bill	54	4

S

B	C	CNT
54	Gold	2
20	Silver	1

ΔS

B	C	CNT
54	Gold	3
23	Silver	1

T

C	D	CNT
Gold	100	2
Gold	80	1
Silver	50	1
Bronze	20	4

$Q(\mathbf{D})$

A	D	CNT
John	100	8
John	80	4
Bill	100	16
Bill	80	8

$\Delta Q(\mathbf{D}, \Delta\mathbf{D})$

A	D	CNT
Mary	50	1

$$\Delta_T Q = \pi_{AD}(R \bowtie \Delta S \bowtie T)$$

$$\equiv \pi_{AD}(\boxed{R \bowtie T} \bowtie \Delta S)$$

Could materialize this, but it is a Cartesian product; doesn't perform better than re-evaluation

Key Insight: conclusion

$Q = \pi_{AD}(R \bowtie S \bowtie T)$

Maintain under all updates

Maintain base tables + query result …

R

A	B	CNT
John	54	2
Mary	23	1
Bill	54	4

S

B	C	CNT
54	Gold	2
20	Silver	1

T

C	D	CNT
Gold	100	2
Gold	80	1
Silver	50	1
Bronze	20	4

$Q(\boldsymbol{D})$

A	D	CNT
John	100	8
John	80	4
Bill	100	16
Bill	80	8

… as well as query subresults as auxiliary views

$\pi_{AC(R \bowtie S)}$

A	C	CNT
John	Gold	4
Bill	Gold	8

$\pi_{BD(S \bowtie T)}$

B	D	CNT
54	100	4
54	80	2
20	Silver	1

Higher-Order Incremental View Maintenance

Higher-Order IVM

Theorem

For a variant of Relational Algebra with Aggregates, Higher-order IVM lowers the complexity of maintenance under single-tuple updates from complexity class AC0/TC0 to complexity class NC0.

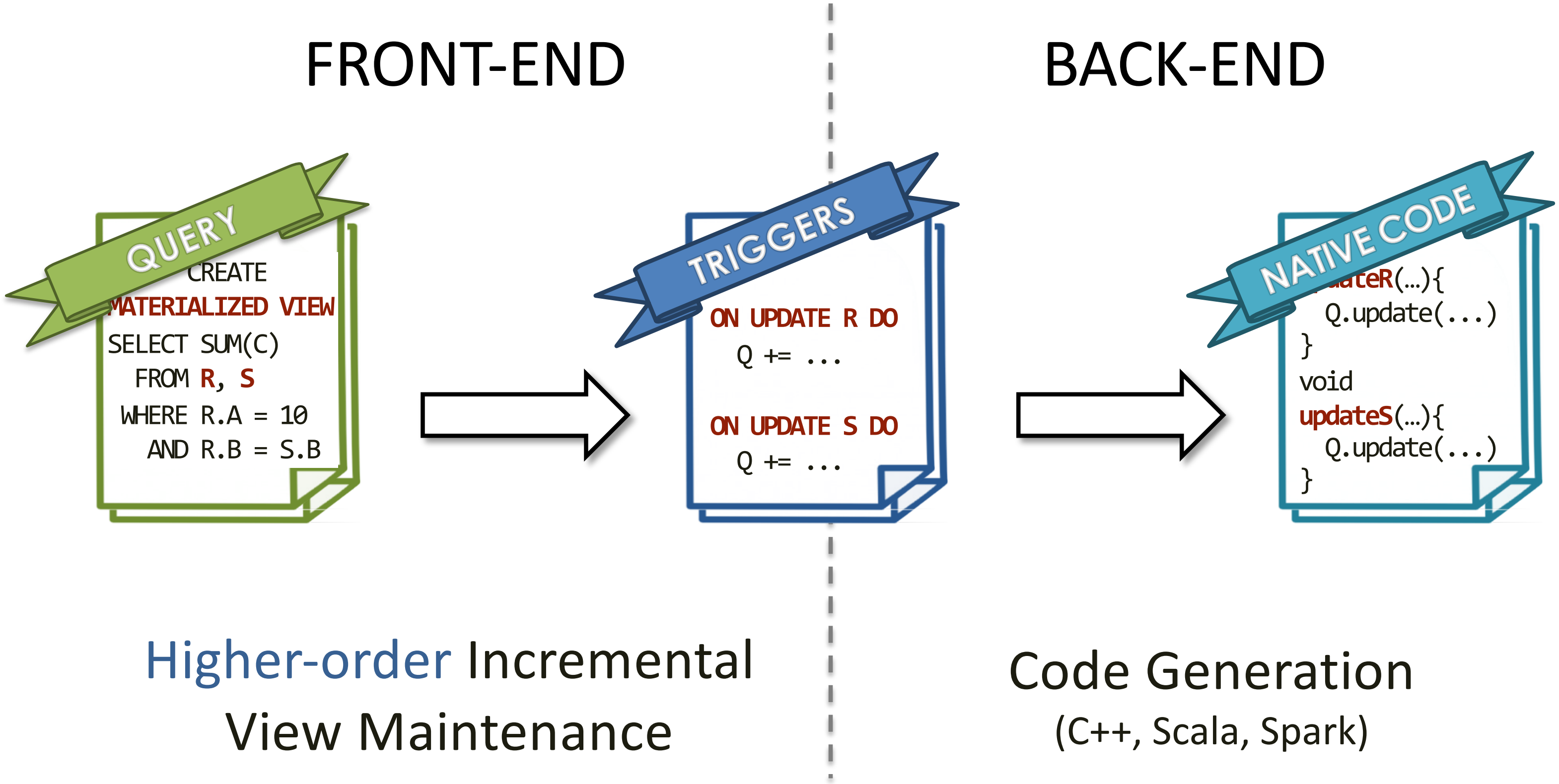
C. Koch. Incremental query evaluation in a ring of databases. PODS 2010

Practical system:

TOASTER **SQL QUERY COMPILER**

DBToaster: higher-order delta processing for dynamic, frequently fresh views. C Koch et al. VLDB J. 23(2), 2014.

TOASTER SQL QUERY COMPILER



FRONT-END

BACK-END

```
CREATE
MATERIALIZED VIEW
SELECT SUM(C)
  FROM R, S
 WHERE R.A = 10
   AND R.B = S.B
```

```
ON UPDATE R DO
  Q += ...

ON UPDATE S DO
  Q += ...
```

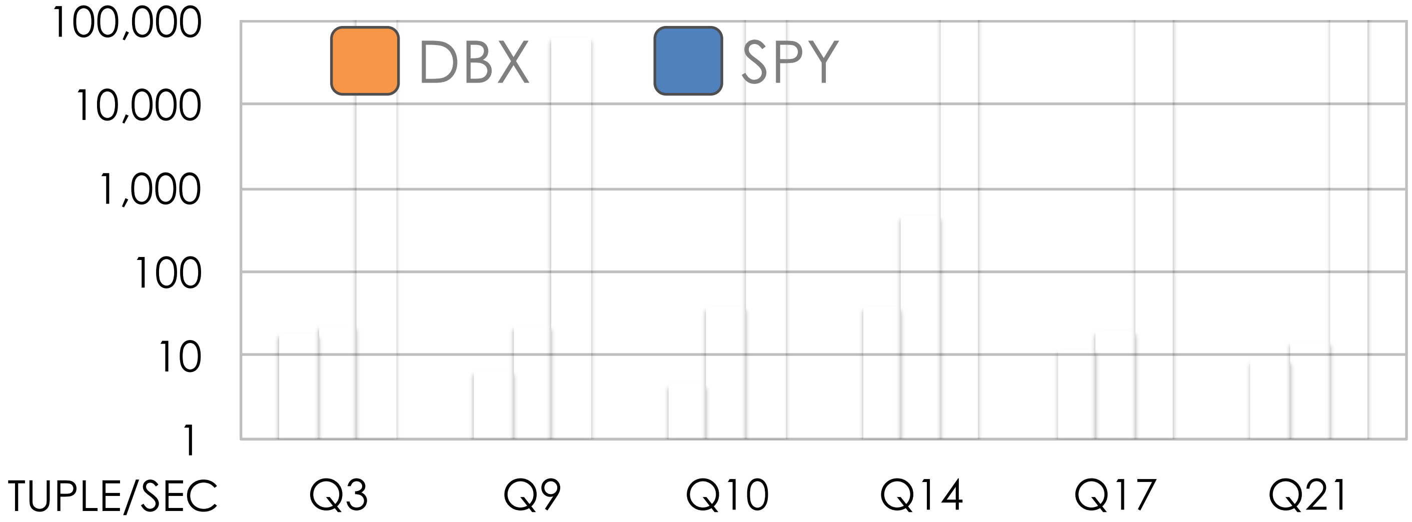
```
ateR(…){
  Q.update(...)
}
void
updateS(…){
  Q.update(...)
}
```

Higher-order Incremental View Maintenance

Code Generation
(C++, Scala, Spark)

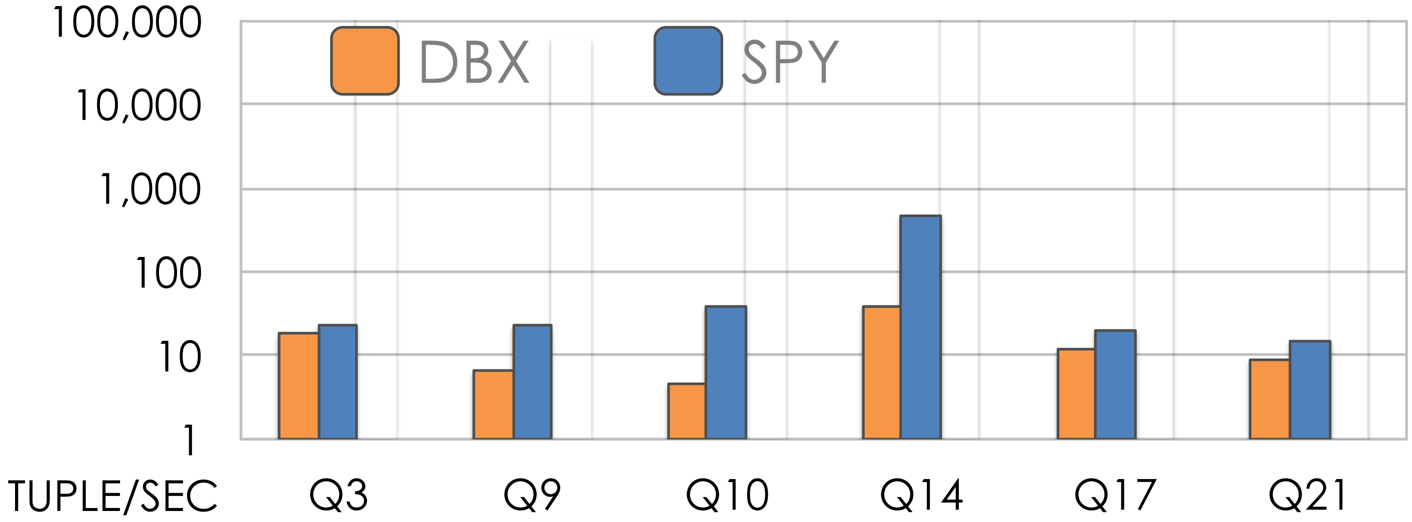
DBToaster: TPC-H BENCHMARK

Single-Tuple Incremental Stream Processing



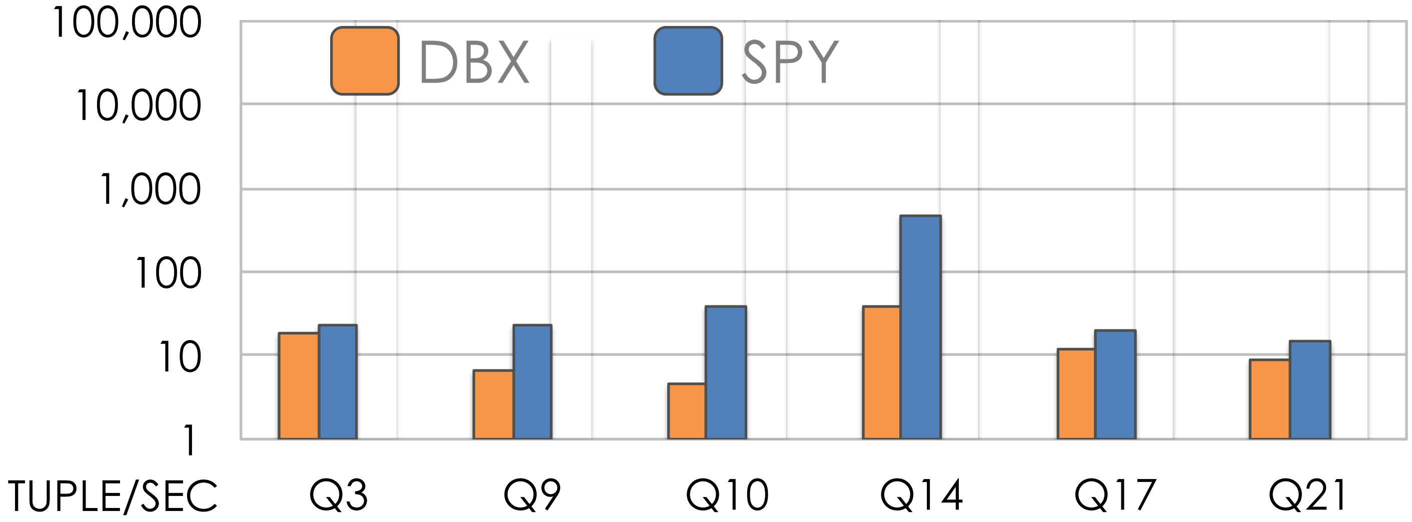
DBToaster: TPC-H BENCHMARK

Single-Tuple Incremental Stream Processing



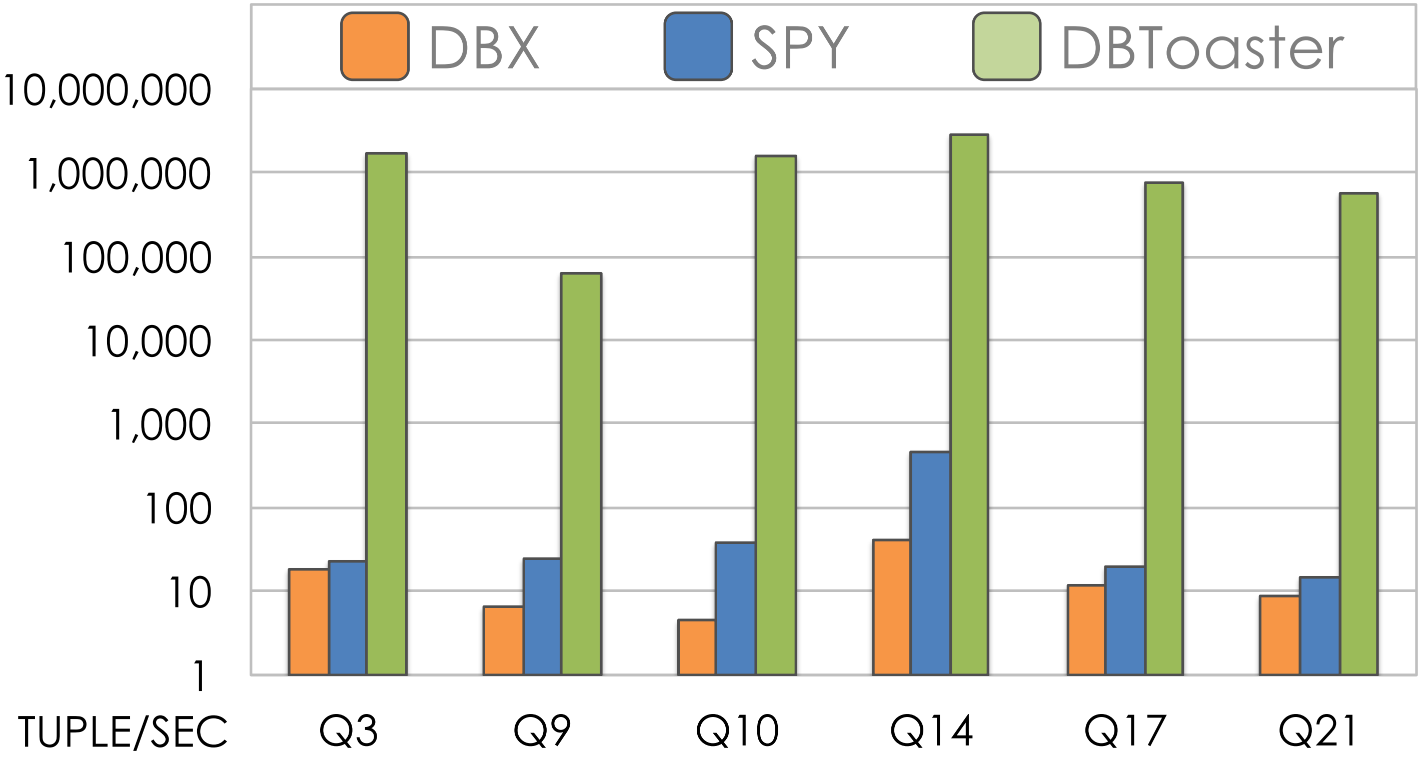
DBToaster: TPC-H BENCHMARK

Single-Tuple Incremental Stream Processing



DBToaster: TPC-H BENCHMARK

Single-Tuple Incremental Stream Processing



HIVM: disadvantage

$Q = \pi_{AD}(R \bowtie S \bowtie T)$

Maintain under all updates

Maintain base tables + query result …

R

A	B	CNT
John	54	2
Mary	23	1
Bill	54	4

S

B	C	CNT
54	Gold	2
20	Silver	1

T

C	D	CNT
Gold	100	2
Gold	80	1
Silver	50	1
Bronze	20	4

$Q(\boldsymbol{D})$

A	D	CNT
John	100	8
John	80	4
Bill	100	16
Bill	80	8

… as well as query subresults as auxiliary views

$\pi_{AC(R \bowtie S)}$

A	C	CNT
John	Gold	4
Bill	Gold	8

$\pi_{BD(S \bowtie T)}$

B	D	CNT
54	100	4
54	80	2
20	Silver	1

👎

Subresults can be of size $|R|\times|S|$ resp. $|S|\times|T|$

In general: can be bigger than |Q(D)|

Not all subresults are useful to materialize

IVM + HIVM: Disadvantage

$$Q = (R \bowtie S \bowtie T)$$

Full join query

(e.g., Complex Event Processing)

Maintain base tables + query result …

R

A	B	CNT
John	54	2
Mary	23	1
Bill	54	4

S

B	C	CNT
54	Gold	2
20	Silver	1

T

C	D	CNT
Gold	100	2
Gold	80	1
Silver	50	1
Bronze	20	4

$Q(\boldsymbol{D})$

A	B	C	D	CNT
John	54	Gold	100	8
John	54	Gold	80	4
Bill	54	Gold	100	16
Bill	54	Gold	80	8

(… as well as query subresults as auxiliary views for HIVM)

- **Memory footprint: $Q(D)$ can be $|D|^2$**
- **Realistic to materialize in-memory ?**

Tradeoff between IVM and HIVM

HIVM

Materialization of Subresults
Memory footprint
(also impacts latency)

↔

IVM

Recomputation of subresults on updates
High update processing latency

Tradeoff between IVM and HIVM

HIVM

Materialization of Subresults

Memory footprint (also impacts latency)

↔

IVM

Recomputation of subresults on updates

High update processing latency

Can we do better?

Key Insight

Avoid storing the materialized views; instead store a compressed representation.

Properties of Materialization of Q(**D**)

$$Q = (R \bowtie S \bowtie T)$$

A	B
1	3
3	4
2	4
2	3

$\bowtie$

B	C
4	3
3	5
6	5
3	2

$\bowtie$

C	D
1	1
3	4
4	5
5	6

=

A	B	C	D
3	4	3	4
2	4	3	4
1	3	5	6
2	3	5	6

↓

Array,
Linked List,
…

👍

Efficient enumeration of Q(DB)

👎

Requires |Q(DB)| space, Ω(N^2) in worst case

Constant-delay enumeration

Enumerate set $Q(D)$with constant delay from a data structure M:

$Q(D)$

#	A	B	C	D
t1	a1	b1	c1	d1
t2	a2	b2	c2	d2
t3	a3	b3	c3	d3
.				
.				
.				
tn	an	bn	cn	dn

Constant time to enumerate first tuple

Constant time between tuples

Constant time between enumerating
last tuple and end of enumeration process

Constant in data complexity: i.e., independent of $|D|$ or $|Q(D)|$,
but may depend on $|Q|$.

Comparable to enumerating from an in-memory array

$\equiv$ Streaming decompression algorithm

Dynamic Yannakakis (DYN)

- Materialize a data structure that is:
  - Succinct**:** no larger than the database **D**
  - From which the query result can be enumerated with constant delay
  - Which can be efficiently maintained under updates

The Dynamic Yannakakis Algorithm:
Compact and Efficient Query Processing Under Updates.
M. Idris, M. Ugarte, S. Vansummeren. SIGMOD 2017

Conjunctive Queries with Inequalities Under Updates.
M. Idris et al. PVLDB 11(7), 2018

Dynamic Yannakakis (DYN)

- Materialize a data structure that is:
  - Succinct**:** no larger than the database **D**
  - From which the query result can be enumerated with constant delay
  - Which can be efficiently maintained under updates

To achieve this, DYN works only on acyclic conjunctive queries
(but extends to deal with aggregation, negation).

Conjunctive Queries (CQs)

Select-project-join queries with equi-joins only

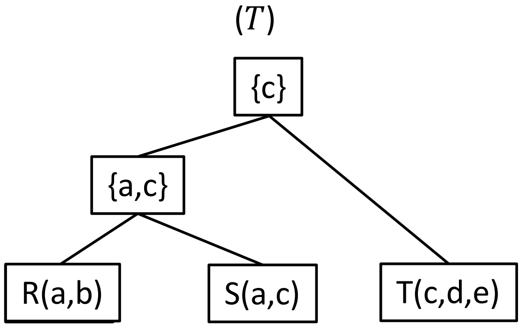
$$Q = \pi_{ac}\left(R(a,b) \bowtie S(a,c) \bowtie T(c,d,e)\right)$$

```
SELECT A, C, SUM(1)
FROM R, S, T
WHERE R.A = S.A and S.C = T.C
GROUP BY A,C
```

Acyclic Queries

A CQ is acyclic if its body admits a Generalized Join Tree (GJT)*

- GJTs for Acyclic CQs: A node labelled tree T where
  - Every leaf is a relation in the query



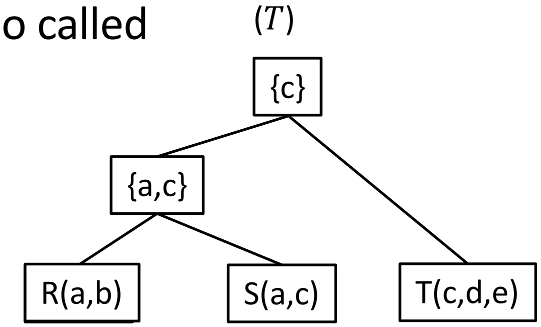
*Does not depend on projections

$$Q = R(a,b) \bowtie S(a,c) \bowtie T(c,d,e)$$

Acyclic Queries

A CQ is acyclic if its body admits a Generalized Join Tree (GJT)*

- GJTs for Acyclic CQs: A node labelled tree T where
  - Every leaf is a relation in the query
  - Internal nodes are sets of attributes that are subset of at least one of its children (also called *guard*)



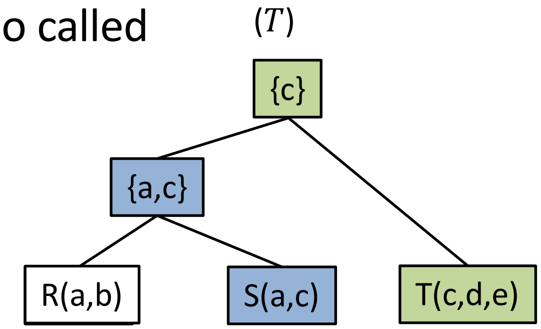
*Does not depend on projections

$$Q = R(a,b) \bowtie S(a,c) \bowtie T(c,d,e)$$

Acyclic Queries

A CQ is acyclic if its body admits a Generalized Join Tree (GJT)*

- GJTs for Acyclic CQs: A node labelled tree T where
  - Every leaf is a relation in the query
  - Internal nodes are sets of attributes that are subset of at least one of its children (also called *guard*)



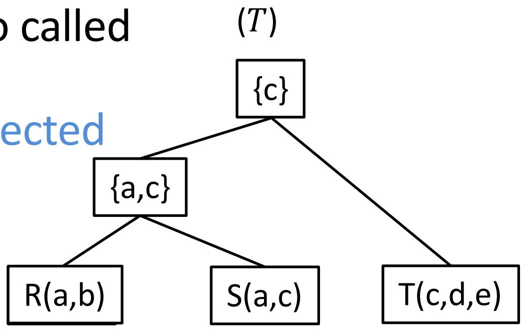
*Does not depend on projections

$$Q = R(a,b) \bowtie S(a,c) \bowtie T(c,d,e)$$

Acyclic Queries

A CQ is acyclic if its body admits a Generalized Join Tree (GJT)*

- GJTs for Acyclic CQs: A node labelled tree T where
  - Every leaf is a relation in the query
  - Internal nodes are sets of attributes that are subset of at least one of its children (also called *guard*)
  - Every variable in the tree induces a connected subtree in T



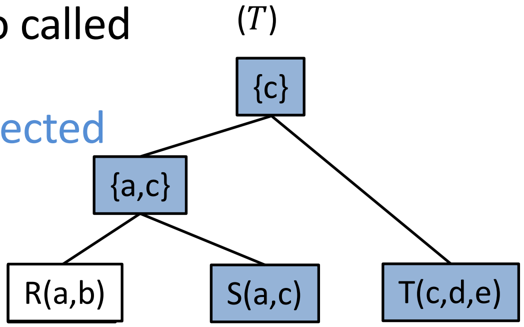
*Does not depend on projections

$$Q = R(a,b) \bowtie S(a,c) \bowtie T(c,d,e)$$

Acyclic Queries

A CQ is acyclic if its body admits a Generalized Join Tree (GJT)*

- GJTs for Acyclic CQs: A node labelled tree T where
  - Every leaf is a relation in the query
  - Internal nodes are sets of attributes that are subset of at least one of its children (also called *guard*)
  - Every variable in the tree induces a connected subtree in T

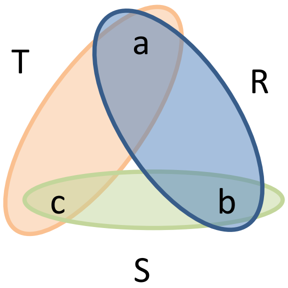


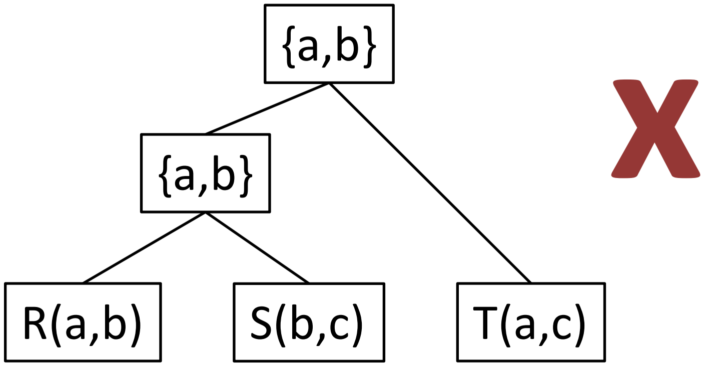
*Does not depend on projections

$$Q = R(a,b) \bowtie S(a,c) \bowtie T(c,d,e)$$

A cyclic query: the triangle query

$$Q = R(a,b) \bowtie S(b,c) \bowtie T(a,c)$$





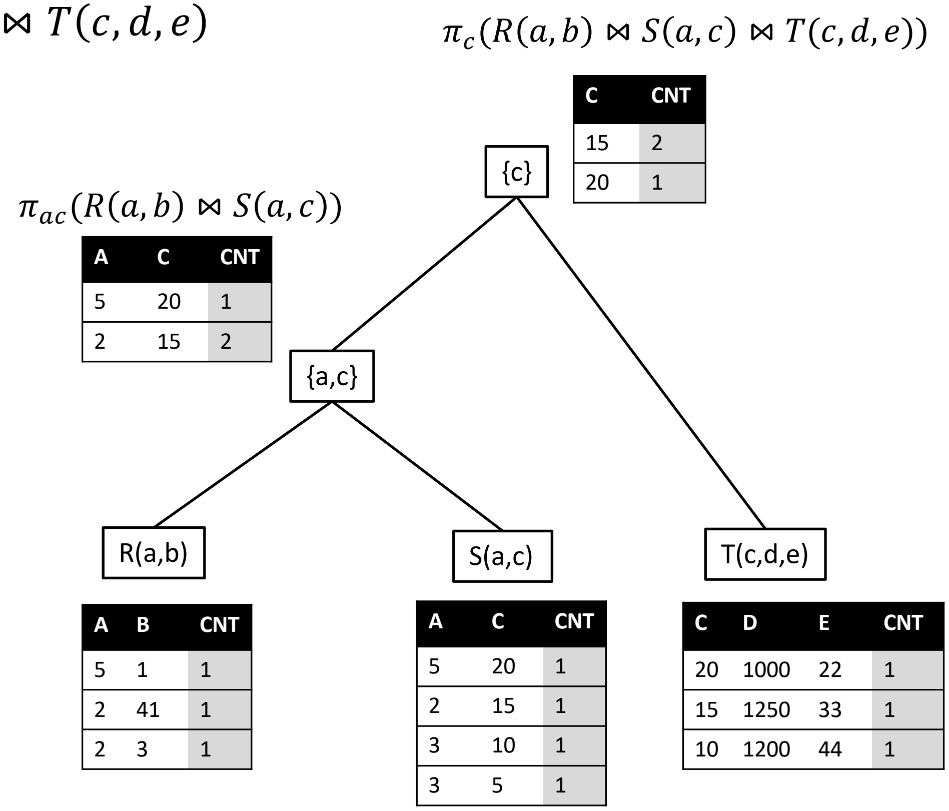
Objectives of Dyanmic Yannakakis (DYN)

- Materialize a data structure that is:
  - Succinct**:** no larger than the database D
  - From which the query result can be enumerated with constant delay
  - Which can be efficiently maintained under updates

T-reduct: Compressed representation based on GJTs

$$Q = R(a,b) \bowtie S(a,c) \bowtie T(c,d,e)$$

- Bottom-up semi-Join reduction
- Linear in the size of database



$\pi_c(R(a,b) \bowtie S(a,c) \bowtie T(c,d,e))$

C	CNT
15	2
20	1

$\pi_{ac}(R(a,b) \bowtie S(a,c))$

A	C	CNT
5	20	1
2	15	2

A	B	CNT
5	1	1
2	41	1
2	3	1

A	C	CNT
5	20	1
2	15	1
3	10	1
3	5	1

C	D	E	CNT
20	1000	22	1
15	1250	33	1
10	1200	44	1

Objectives of Dyanmic Yannakakis (DYN)

- Materialize a data structure that is:
  - Succinct**:** no larger than the database **D**
  - From which the query result can be enumerated with constant delay
  - Which can be efficiently maintained under updates

T-reduct: Enumeration

$$Q = R(a,b) \bowtie S(a,c) \bowtie T(c,d,e)$$

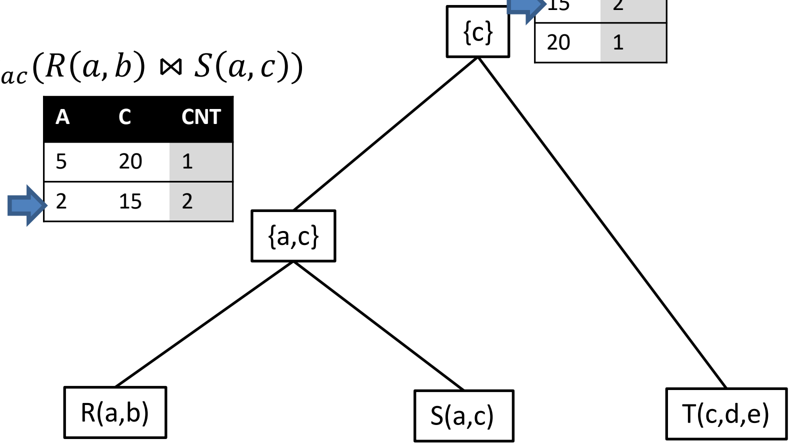
- Top-down: find "compatible" tuples

$\pi_c(R(a,b) \bowtie S(a,c) \bowtie T(c,d,e))$

C	CNT
15	2
20	1

$\pi_{ac}(R(a,b) \bowtie S(a,c))$

A	C	CNT
5	20	1
2	15	2



A	B	CNT
5	1	1
2	41	1
2	3	1

A	C	CNT
5	20	1
2	15	1
3	10	1
3	5	1

C	D	E	CNT
20	1000	22	1
15	1250	33	1
10	1200	44	1

Output:

A	B	C	D	E	CNT
2	41	15	1250	33	1

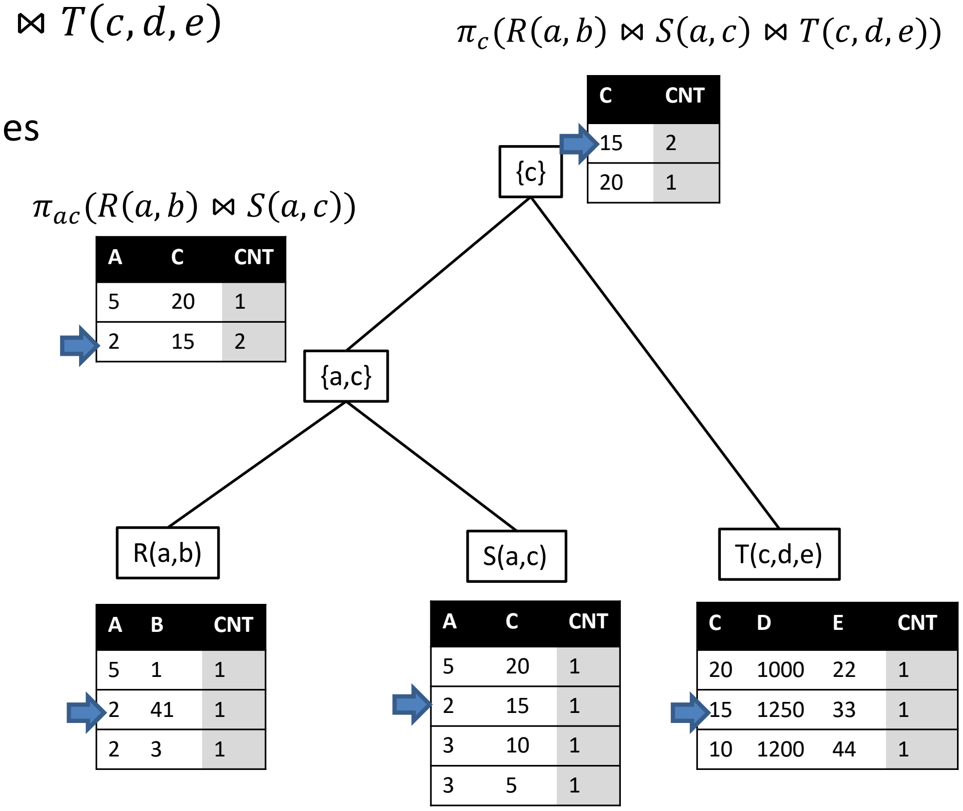
T-reduct: Enumeration

$$Q = R(a,b) \bowtie S(a,c) \bowtie T(c,d,e)$$

- Top-down: find “compatible” tuples

Output:

A	B	C	D	E	CNT
2	41	15	1250	33	1
2	3	15	1250	33	1



$\pi_c(R(a,b) \bowtie S(a,c) \bowtie T(c,d,e))$

C	CNT
15	2
20	1

$\pi_{ac}(R(a,b) \bowtie S(a,c))$

A	C	CNT
5	20	1
2	15	2

A	B	CNT
5	1	1
2	41	1
2	3	1

A	C	CNT
5	20	1
2	15	1
3	10	1
3	5	1

C	D	E	CNT
20	1000	22	1
15	1250	33	1
10	1200	44	1

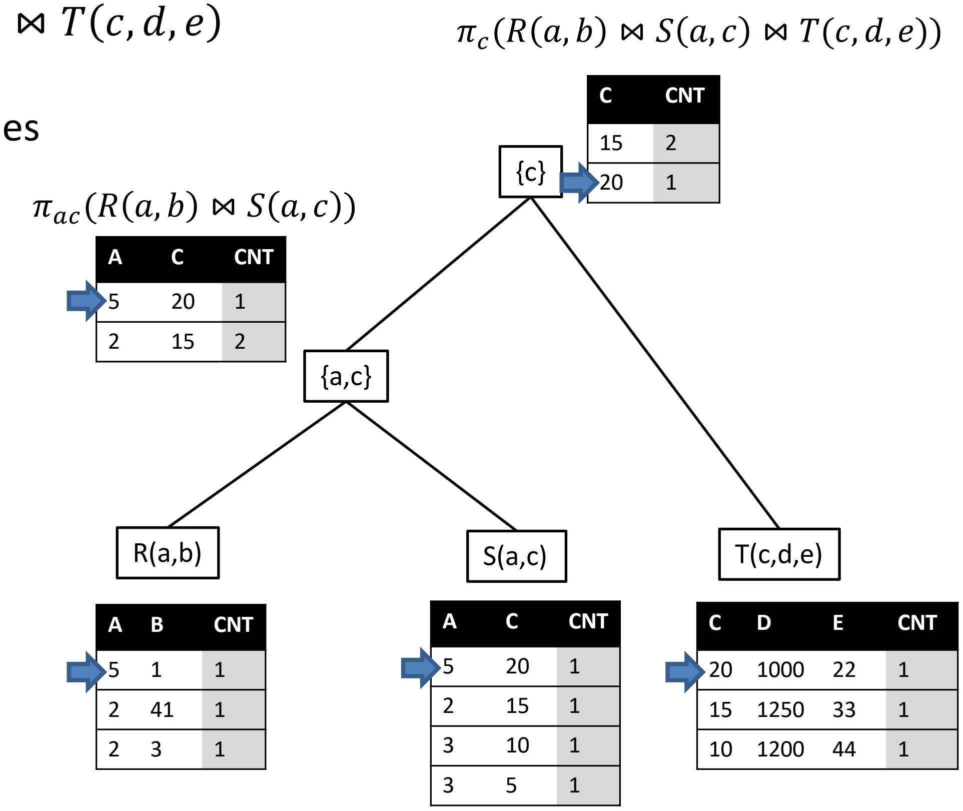
T-reduct: Enumeration

$$Q = R(a,b) \bowtie S(a,c) \bowtie T(c,d,e)$$

- Top-down: find “compatible” tuples

Output:

A	B	C	D	E	CNT
2	**41**	**15**	**1250**	**33**	1
2	**3**	**15**	**1250**	**33**	1
5	1	20	1000	22	1



$\pi_c(R(a,b) \bowtie S(a,c) \bowtie T(c,d,e))$

C	CNT
15	2
20	1

$\pi_{ac}(R(a,b) \bowtie S(a,c))$

A	C	CNT
5	20	1
2	15	2

A	B	CNT
5	1	1
2	41	1
2	3	1

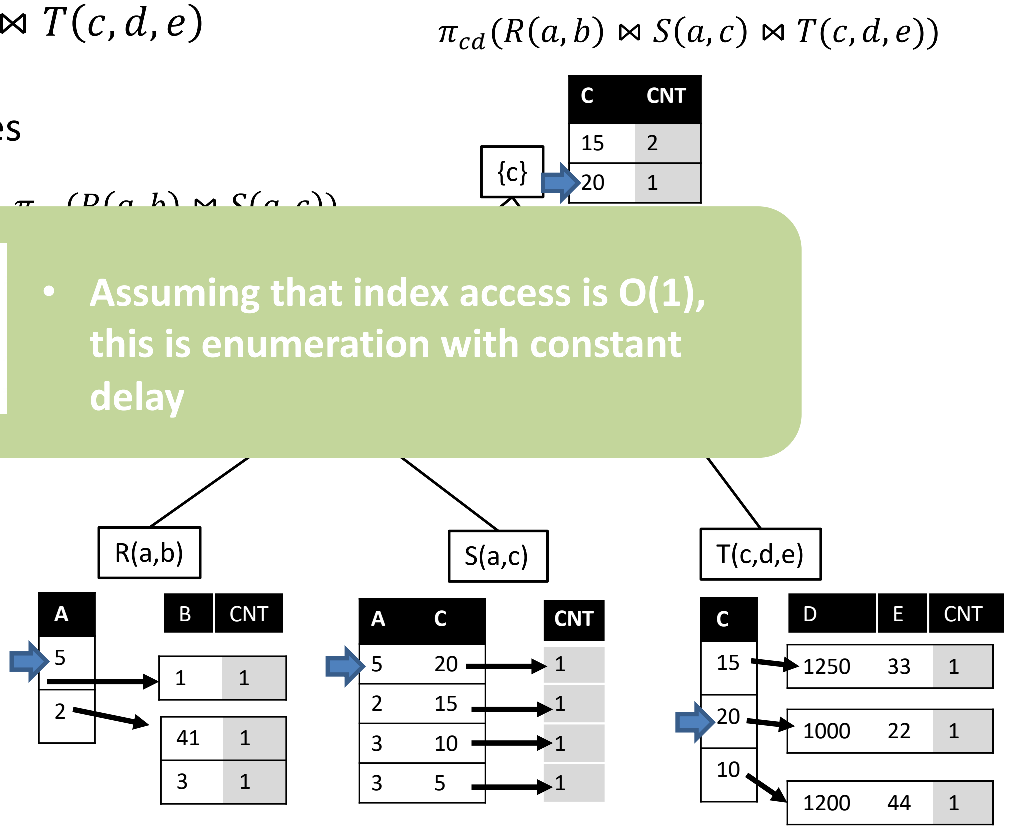
A	C	CNT
5	20	1
2	15	1
3	10	1
3	5	1

C	D	E	CNT
20	1000	22	1
15	1250	33	1
10	1200	44	1

T-reduct: Enumeration

$$Q = R(a,b) \bowtie S(a,c) \bowtie T(c,d,e)$$

- Top-down: find “compatible” tuples
- Add indexes to make this efficient



$\pi_{cd}(R(a,b) \bowtie S(a,c) \bowtie T(c,d,e))$
{c}
$\pi_{..}(R(a,b) \bowtie S(a,c))$
R(a,b)
S(a,c)
T(c,d,e)

C	CNT
15	2
20	1

R(a,b):

A
5
2

B	CNT
1	1
41	1
3	1

S(a,c):

A	C	CNT
5	20	1
2	15	1
3	10	1
3	5	1

T(c,d,e):

C
15
20
10

D	E	CNT
1250	33	1
1000	22	1
1200	44	1

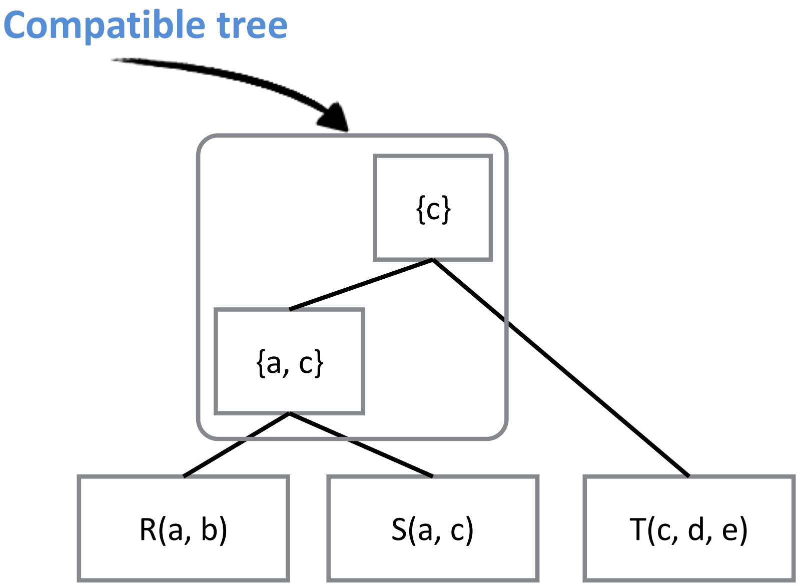
- Assuming that index access is O(1), this is enumeration with constant delay

Output:

A	B	C	D	E	CNT
2	41	15	1250	33	1
2	3	15	1250	33	1
5	1	20	1000	22	1

What about projections ?

$$Q = \pi_{a,c}\big(R(a,b) \bowtie S(a,c) \bowtie T(c,d,e)\big)$$



We can still enumerate
with constant delay!

Objectives of Dyanmic Yannakakis (DYN)

- Materialize a data structure that is:
  - Succinct**:** no larger than the database D
  - From which the query result can be enumerated with constant delay
  - Which can be efficiently maintained under updates

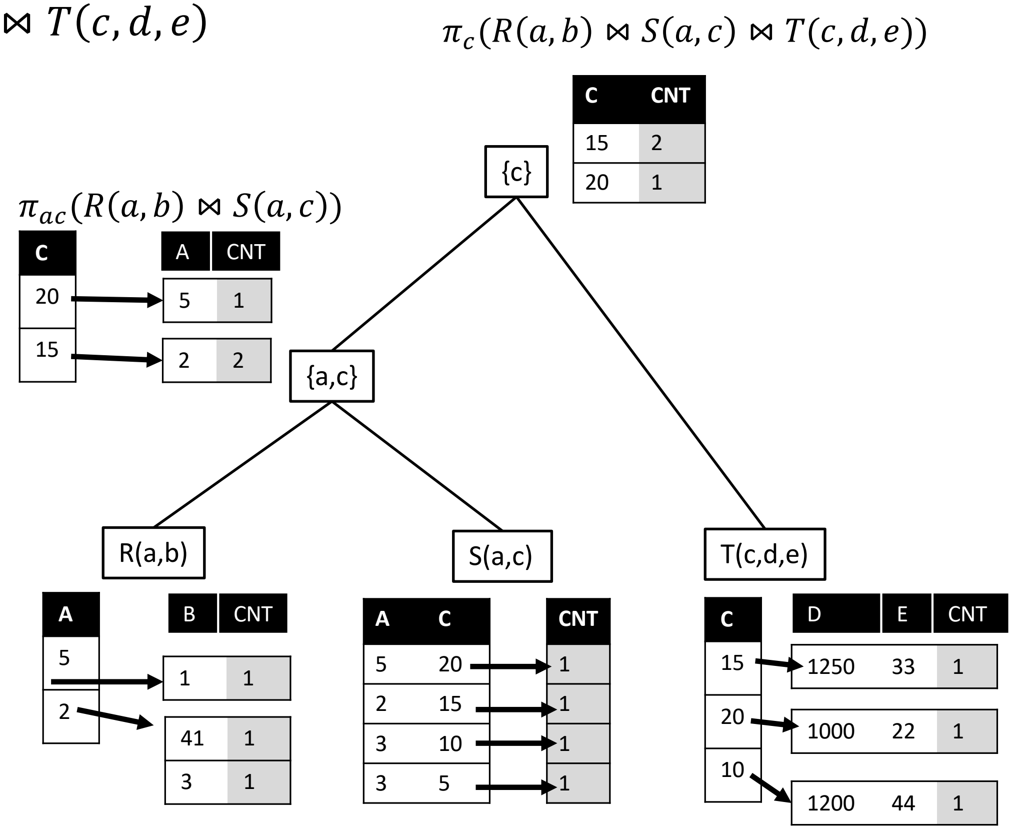
T-reduct: Update processing

$$Q = R(a,b) \bowtie S(a,c) \bowtie T(c,d,e)$$

- Bottom-up: propagate update through semi-join reduction (view each node as IVM)

ΔS

A	C	CNT
5	10	1
5	20	-1



$\pi_c(R(a,b) \bowtie S(a,c) \bowtie T(c,d,e))$

C	CNT
15	2
20	1

$\pi_{ac}(R(a,b) \bowtie S(a,c))$

C	A	CNT
20	5	1
15	2	2

A	B	CNT
5	1	1
2	41	1
	3	1

A	C	CNT
5	20	1
2	15	1
3	10	1
3	5	1

C	D	E	CNT
15	1250	33	1
20	1000	22	1
10	1200	44	1

T-reduct: Update processing

$$Q = R(a,b) \bowtie S(a,c) \bowtie T(c,d,e)$$

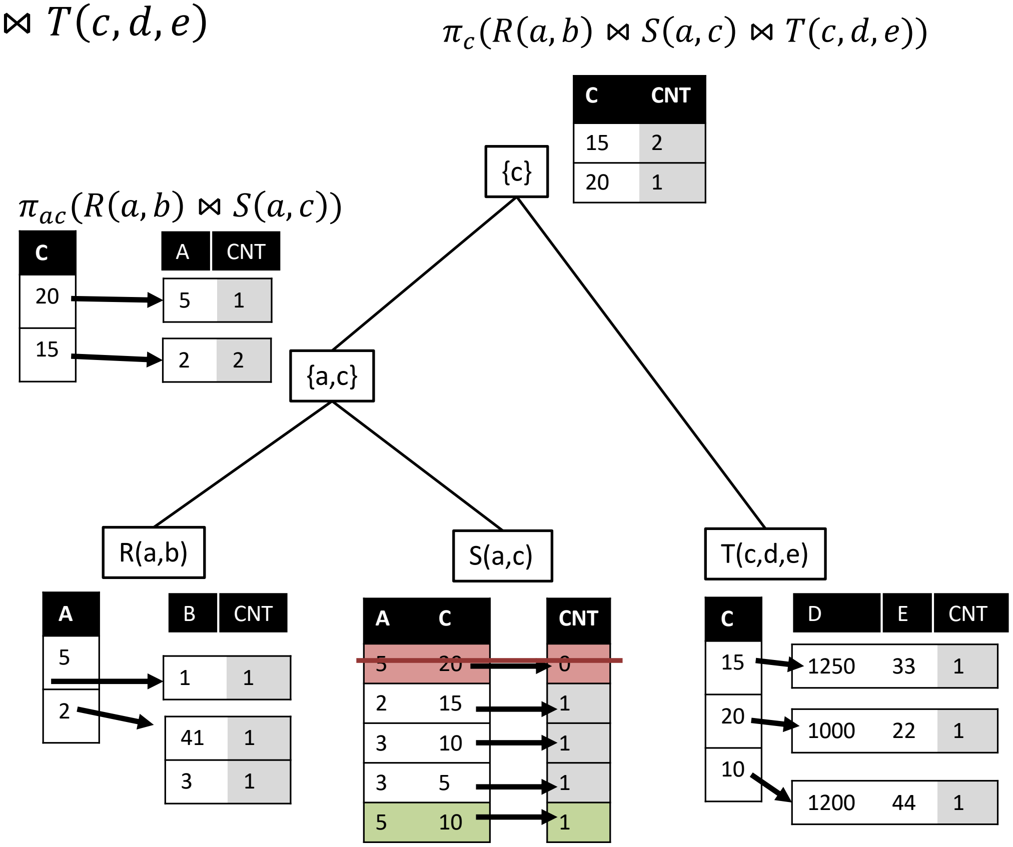
- Bottom-up: propagate update through semi-join reduction (view each node as IVM)

$\Delta_{ac=} \pi_{ac}(R(a,b) \bowtie \Delta S(a,c))$

A	C	CNT
5	10	1
5	20	-1

ΔS

A	C	CNT
5	10	1
5	20	-1



$\pi_c(R(a,b) \bowtie S(a,c) \bowtie T(c,d,e))$

C	CNT
15	2
20	1

$\pi_{ac}(R(a,b) \bowtie S(a,c))$

C	A	CNT
20	5	1
15	2	2

A	B	CNT
5	1	1
2	41	1
	3	1

A	C	CNT
~~5~~	~~20~~	~~0~~
2	15	1
3	10	1
3	5	1
5	10	1

C	D	E	CNT
15	1250	33	1
20	1000	22	1
10	1200	44	1

T-reduct: Update processing

$$Q = R(a,b) \bowtie S(a,c) \bowtie T(c,d,e)$$

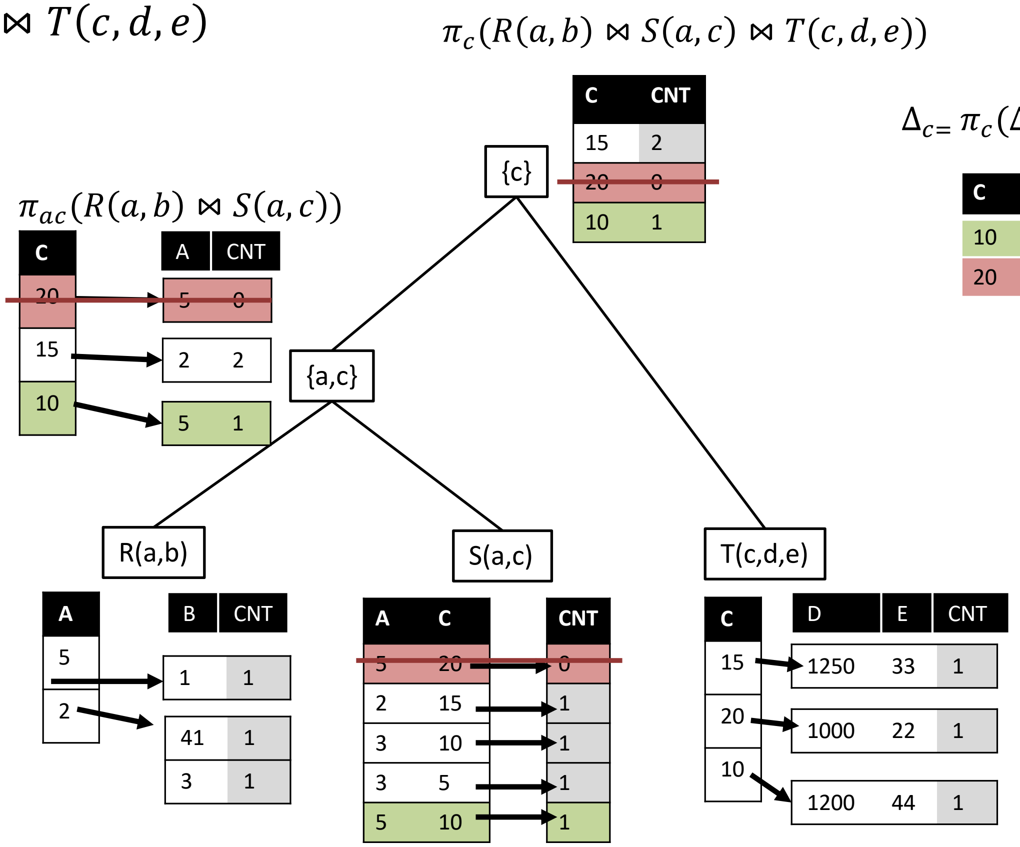
- Bottom-up: propagate update through semi-join reduction (view each node as IVM)

$\Delta_{ac=}\, \pi_{ac}(R(a,b) \bowtie \Delta S(a,c))$

A	C	CNT
5	10	1
5	20	-1

ΔS

A	C	CNT
5	10	1
5	20	-1



$\pi_c(R(a,b) \bowtie S(a,c) \bowtie T(c,d,e))$

C	CNT
15	2
~~20~~	~~0~~
10	1

$\pi_{ac}(R(a,b) \bowtie S(a,c))$

C	A	CNT
~~20~~	~~5~~	~~0~~
15	2	2
10	5	1

A	B	CNT
5	1	1
2	41	1
	3	1

A	C	CNT
~~5~~	~~20~~	~~0~~
2	15	1
3	10	1
3	5	1
5	10	1

C	D	E	CNT
15	1250	33	1
20	1000	22	1
10	1200	44	1

$\Delta_{c=}\, \pi_c(\Delta_{ac} \bowtie T(c,d,e))$

C	CNT
10	1
20	-1

T-reduct: Update processing

$$Q = \ R(a,b) \bowtie S(a,c) \bowtie T(c,d,e)$$

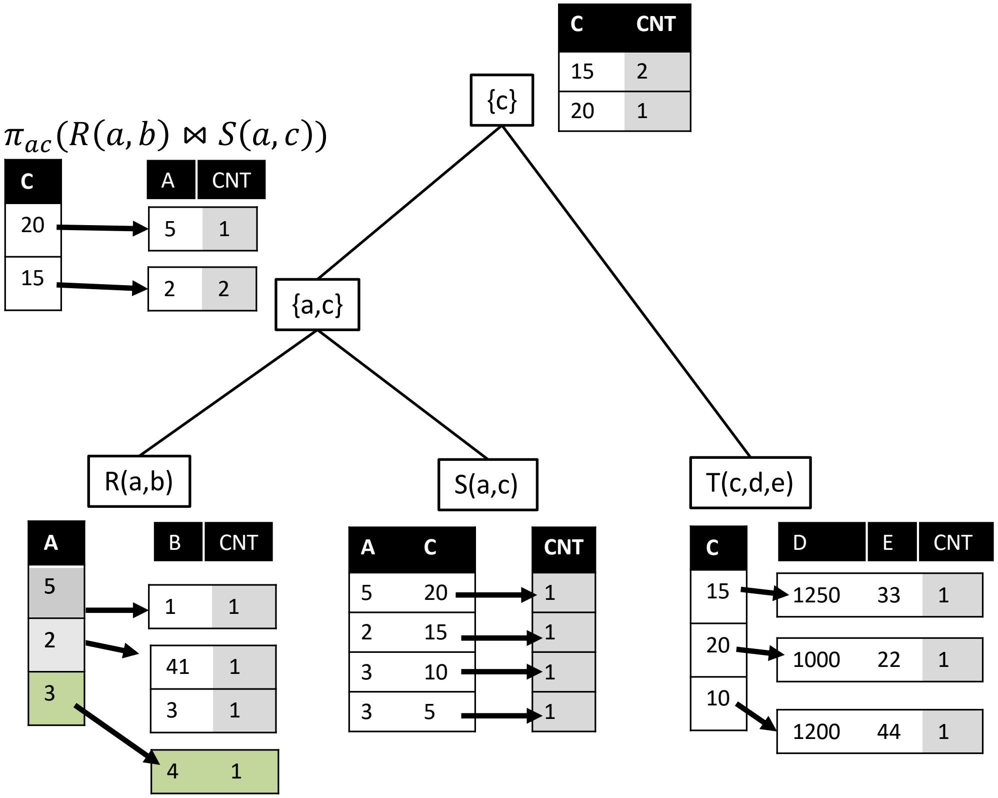
- Index guards on attributes shared with sibling

$$\Delta_{ac=}\, \pi_{ac}(\Delta R(a,b) \bowtie S(a,c))$$

A	C	CNT
3	10	1
3	5	1

ΔR

A	B	CNT
3	4	1



$\pi_c(R(a,b) \bowtie S(a,c) \bowtie T(c,d,e))$

C	CNT
15	2
20	1

$\pi_{ac}(R(a,b) \bowtie S(a,c))$

C	A	CNT
20	5	1
15	2	2

R(a,b)

A	B	CNT
5	1	1
2	41	1
	3	1
3	4	1

S(a,c)

A	C	CNT
5	20	1
2	15	1
3	10	1
3	5	1

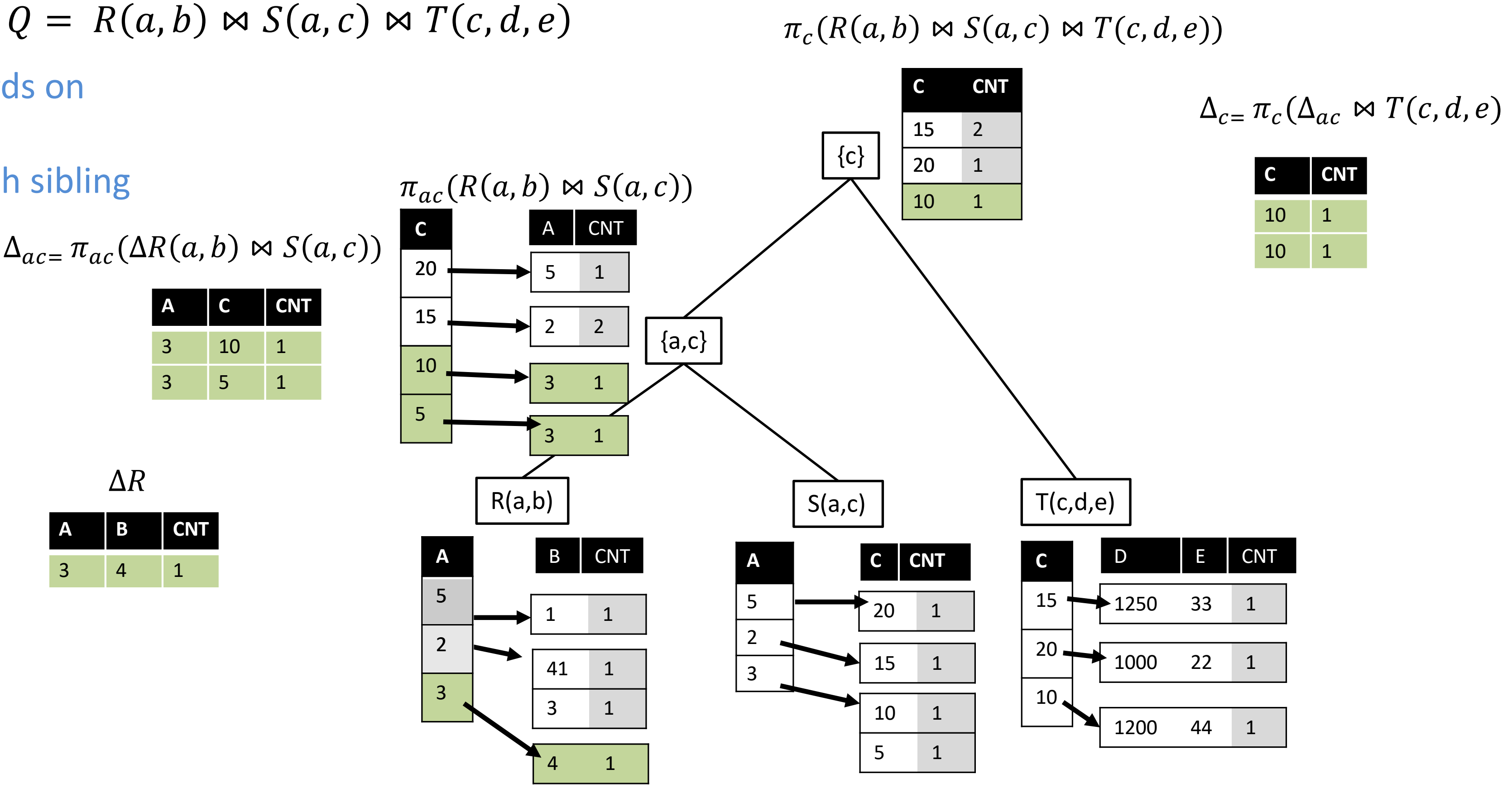
T(c,d,e)

C	D	E	CNT
15	1250	33	1
20	1000	22	1
10	1200	44	1

T-reduct: Update processing

$$Q = R(a,b) \bowtie S(a,c) \bowtie T(c,d,e)$$

- Index guards on attributes shared with sibling



$\pi_c(R(a,b) \bowtie S(a,c) \bowtie T(c,d,e))$

C	CNT
15	2
20	1
10	1

$\Delta_{c=} \pi_c(\Delta_{ac} \bowtie T(c,d,e)$

C	CNT
10	1
10	1

$\pi_{ac}(R(a,b) \bowtie S(a,c))$

C	A	CNT
20	5	1
15	2	2
10	3	1
5	3	1

$\Delta_{ac=} \pi_{ac}(\Delta R(a,b) \bowtie S(a,c))$

A	C	CNT
3	10	1
3	5	1

ΔR

A	B	CNT
3	4	1

R(a,b)

A	B	CNT
5	1	1
2	41	1
	3	1
3	4	1

S(a,c)

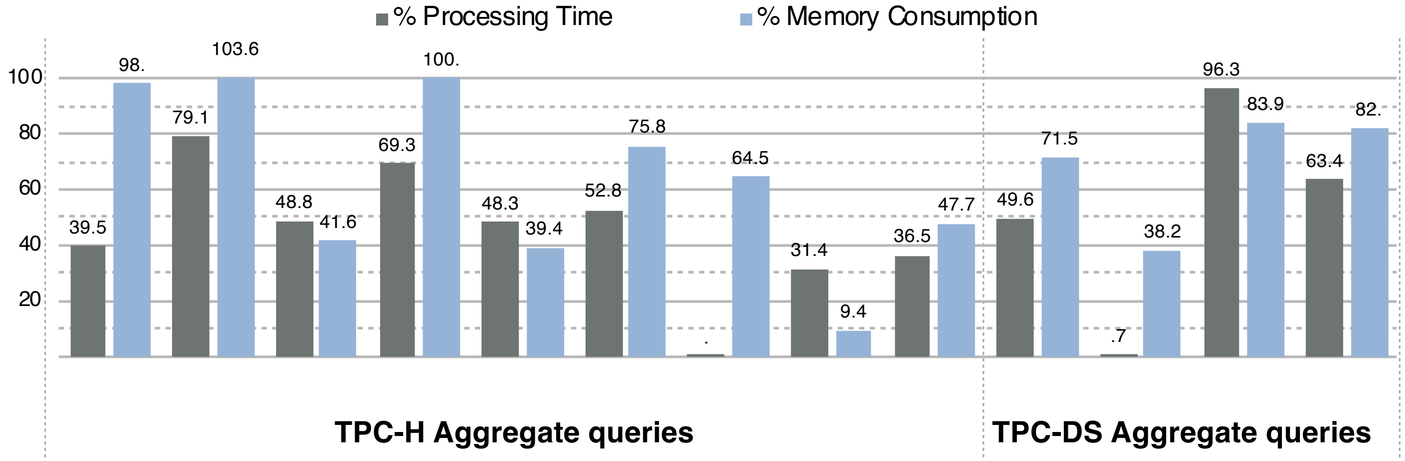
A	C	CNT
5	20	1
2	15	1
3	10	1
	5	1

T(c,d,e)

C	D	E	CNT
15	1250	33	1
20	1000	22	1
10	1200	44	1

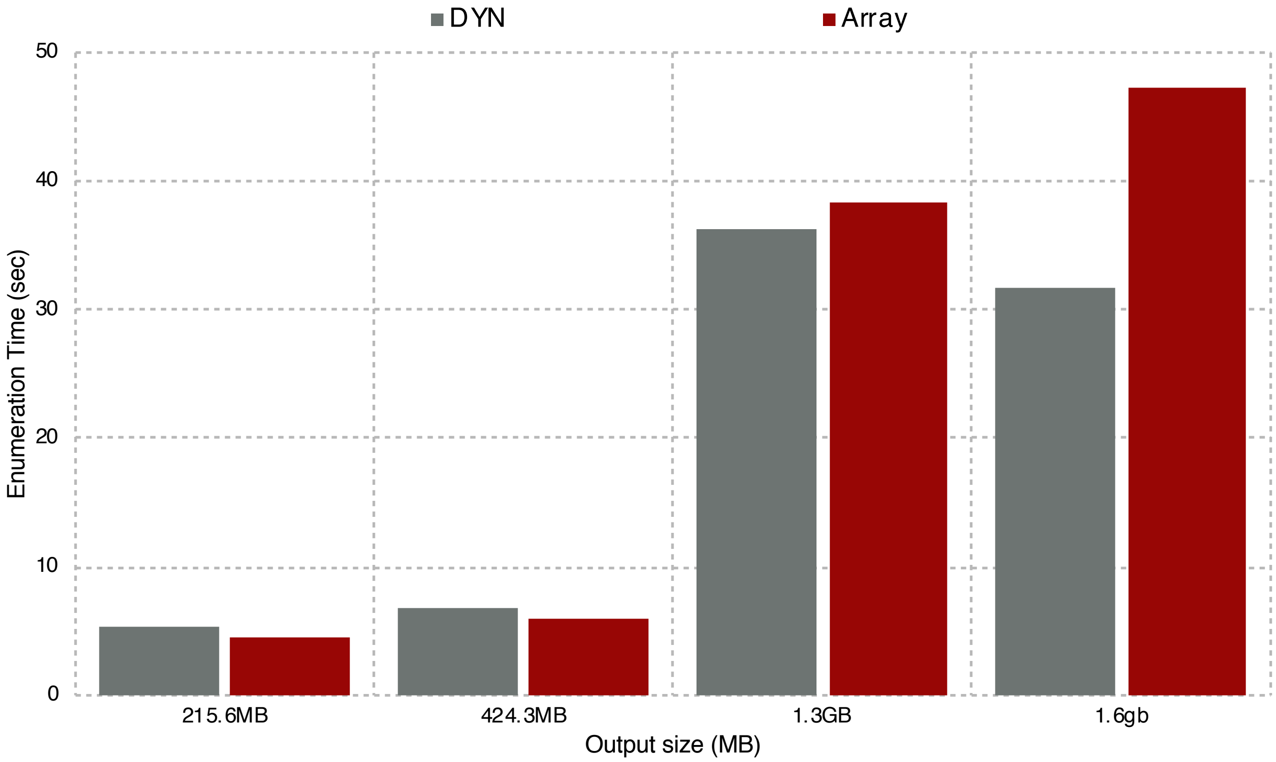
DYN in practice

100% = DBToaster



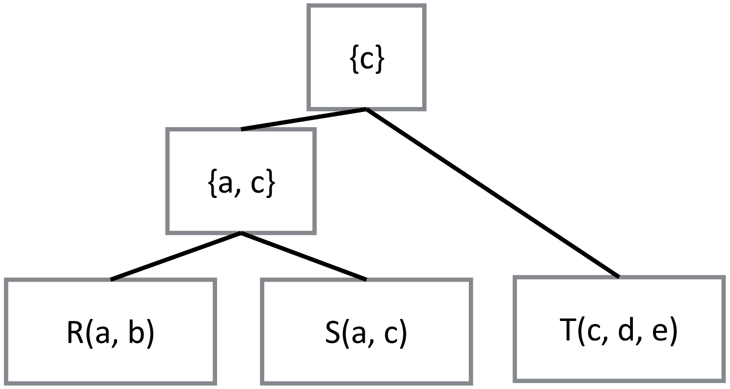
- Up to two order of magnitude faster
- Consumes up to one order of magnitude less memory

Enumeration

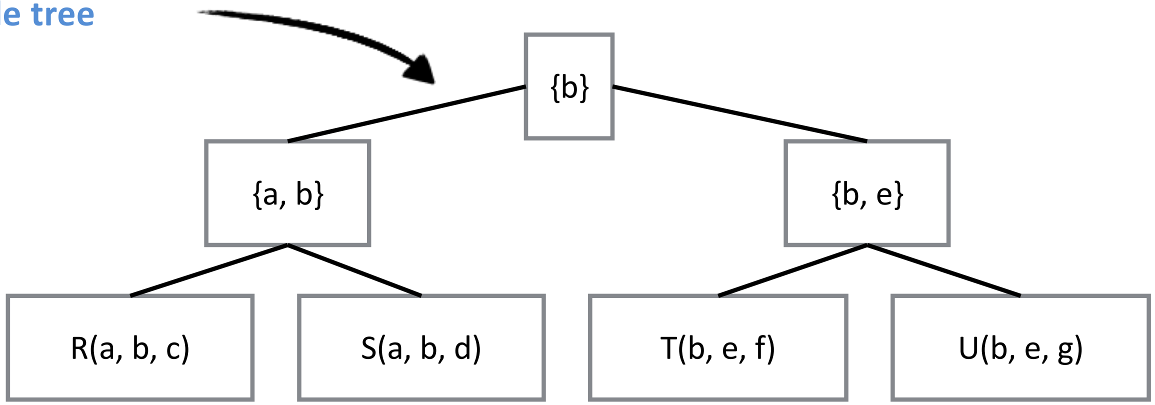


A note about complexity

Assume $|\Delta D|$ is constant, is the required update time also constant?



Simple tree



A note about complexity

- Ideal IVM algorithm allows, for any query Q
  - Constant delay enumeration of $Q(D)$
  - Constant-time update processing if $|\Delta D|$ is constant

[Berkholz et al., PODS 2017; ICDT 2018]

Conjunctive query Q supports constant-delay enumeration after constant-time updates

↔

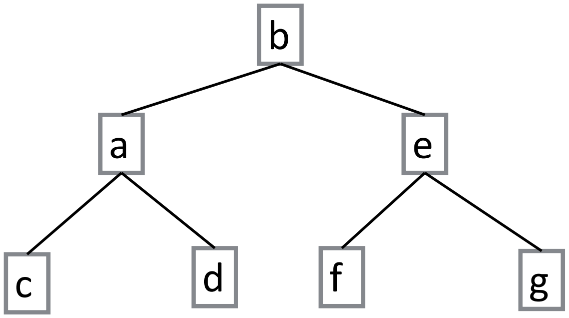
Q is *q-hierarchical*

(under certain complexity-theoretical assumptions)

Q-hierarchical CQs

A CQ is **q-hierarchical** if its body has a **q-tree***

$$\pi_{a,b,e}(R(a,b,c) \bowtie S(a,b,d) \bowtie T(b,e,f) \bowtie U(b,e,g))$$



Atom condition: each atom must induce a directed path, starting from the root

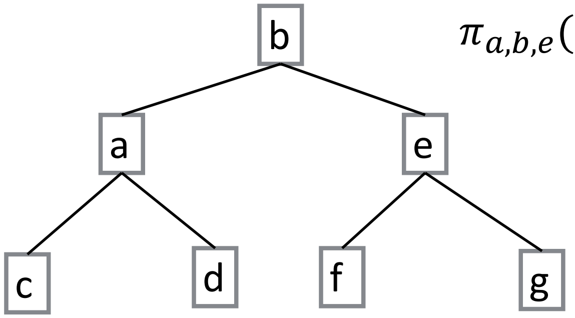
Projection condition: the projection attributes must induce a connected subtree that contains the root

*Does depend on projections

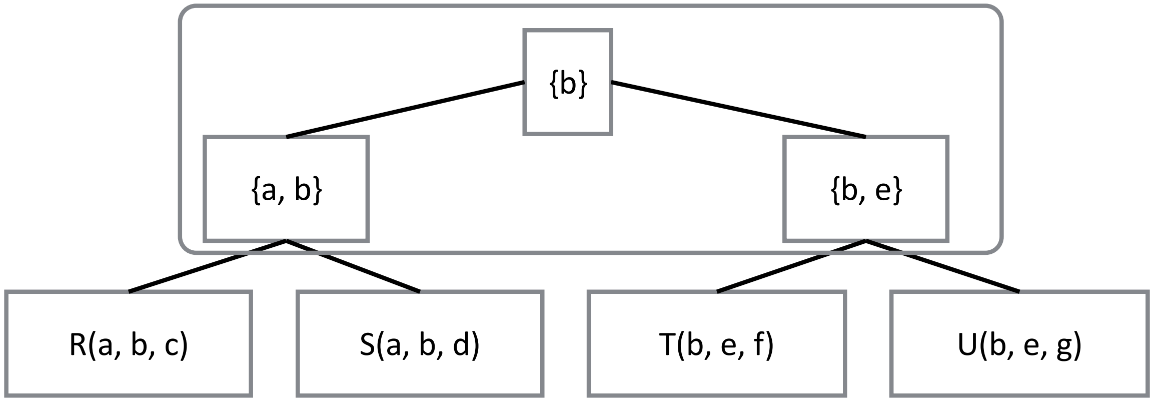
Q-hierarchical CQs

Theorem

A CQ is q-hierarchical if and only if it has a generalized join tree that is both simple and compatible with the projection



$$\pi_{a,b,e}(R(a,b,c) \bowtie S(a,b,d) \bowtie T(b,e,f) \bowtie U(b,e,g))$$



Q-hierarchical CQs

Theorem

A CQ is q-hierarchical if and only if it has a generalized join tree that is both simple and compatible with the projection

Corollary

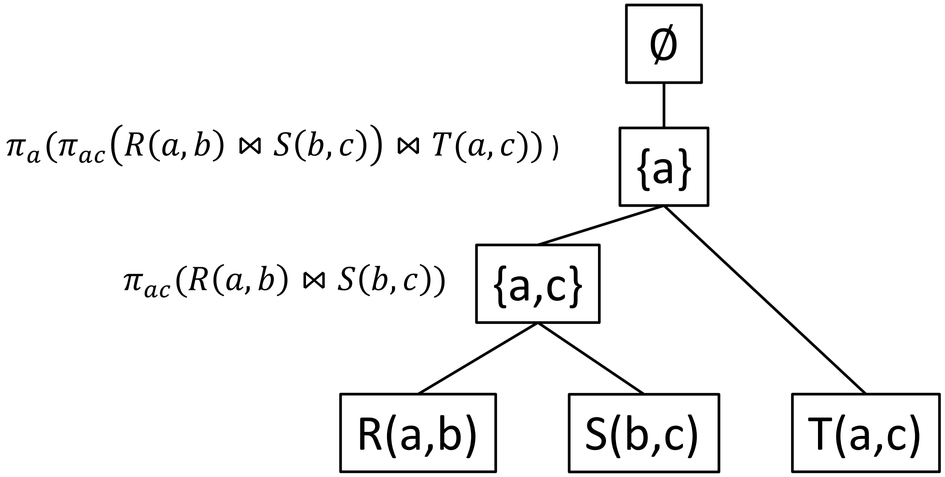
DYN provides constant-delay enumeration after constant-time updates precisely for the class of q-hierarchical queries.

Matches the theoretical lower bound. ✓

What about Cyclic Queries?

- Use View Trees, which relax constraints on GJTs:
  - interior nodes need not have guards
  - Give up connectedness condition

$$Q = \pi_{()}(R(a,b) \bowtie S(b,c) \bowtie T(a,c))$$



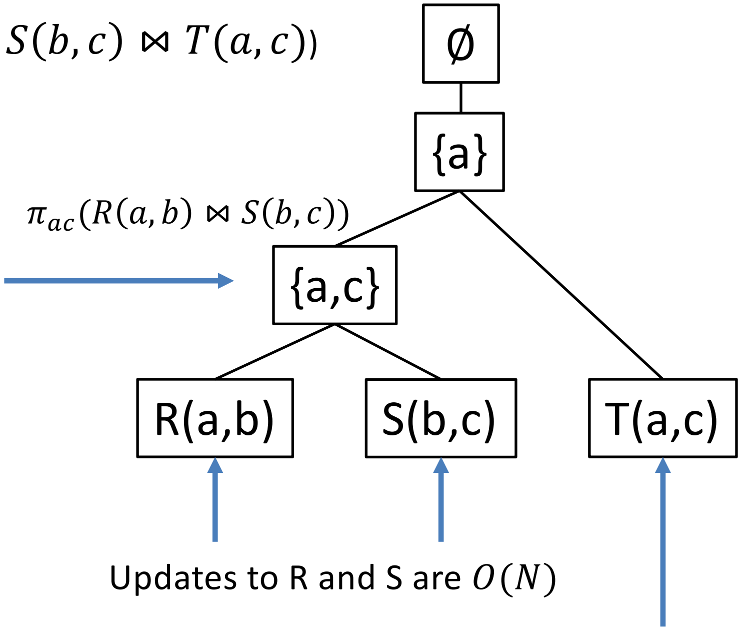
$\pi_a(\pi_{ac}\big(R(a,b) \bowtie S(b,c)\big) \bowtie T(a,c))\,)$

$\pi_{ac}(R(a,b) \bowtie S(b,c))$

What about Cyclic Queries?

- Use View Trees, relaxing constraints on GJTs
- Nodes without guard may be superlinear in $|\boldsymbol{D}|$ but may help processing to some updates

$$Q = \pi_{()}(R(a,b) \bowtie S(b,c) \bowtie T(a,c))$$



$\pi_{ac}(R(a,b) \bowtie S(b,c))$

Size $\Omega(N^2)$ in worst case.

Updates to R and S are $O(N)$

Updates to T are constant

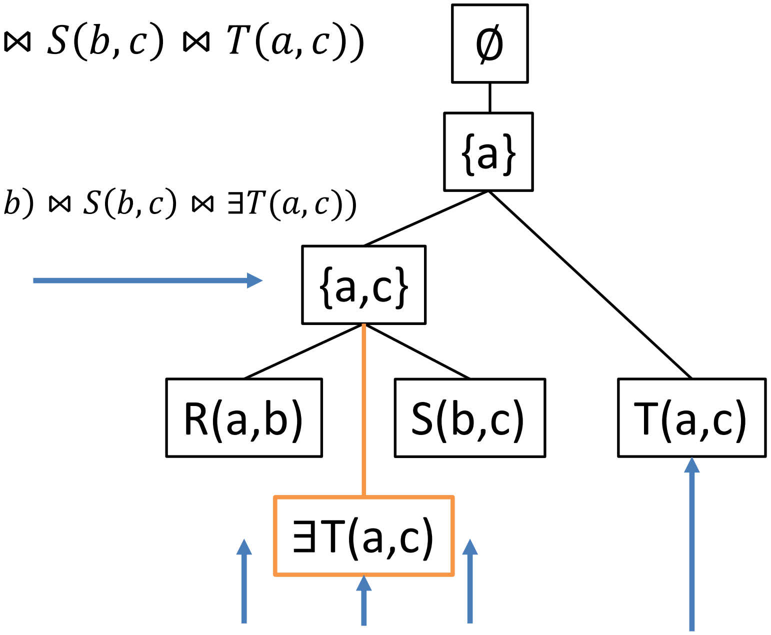
Assumptions:

- $|R| = |S| = |T| = N$
- $|\Delta D|$ = constant

What about Cyclic Queries?

- Use View Trees, relaxing constraints on GJTs
- Nodes without guard may be superlinear in $|\boldsymbol{D}|$ but may help processing to some updates
- Add indicator projections to minimize space

$$Q = \pi_{()}(R(a,b) \bowtie S(b,c) \bowtie T(a,c))$$



$\pi_{ac}(R(a,b) \bowtie S(b,c) \bowtie \exists T(a,c))$

Size $\Omega(N)$.

Updates to R and S are $O(N)$

Updates to T that do not modify $\exists T(a,c)$ are constant, $O(N)$ otherwise

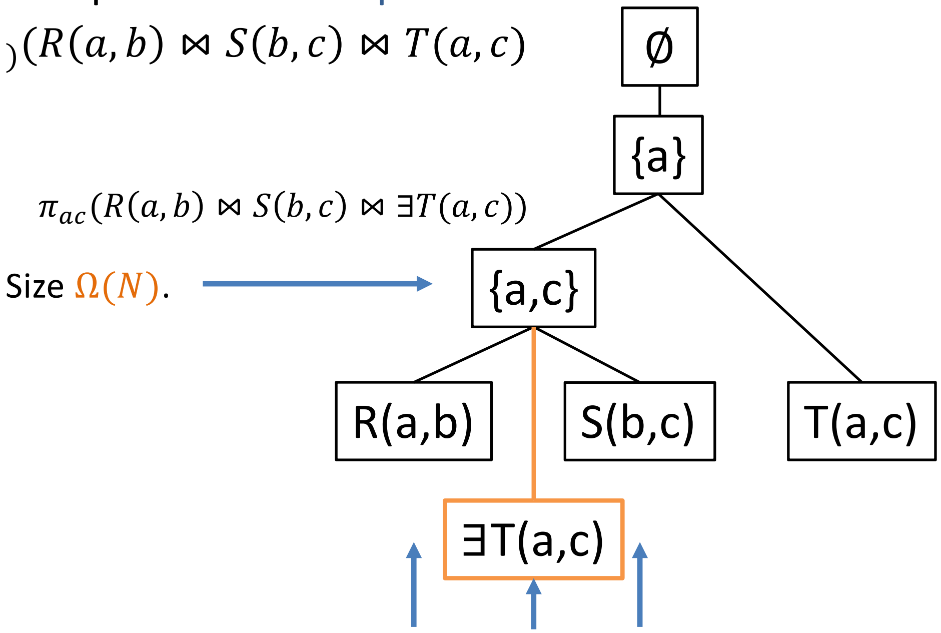
Assumptions:

- $|R| = |S| = |T| = N$
- $|\Delta D|$ = constant

What about Cyclic Queries?

- Use View Trees, relaxing constraints on GJTs
- Nodes without guard may be superlinear in $|\boldsymbol{D}|$ but may help processing to some updates
- Add indicator projections to minimize space and bulk update time

$$Q = \pi_{()}(R(a,b) \bowtie S(b,c) \bowtie T(a,c))$$



$\pi_{ac}(R(a,b) \bowtie S(b,c) \bowtie \exists T(a,c))$

Size $\Omega(N)$.

Assumptions:

- $|R| = |S| = |T| = N$
- Bulk updates, $|\Delta D| =$ N

Bulk updates to R, S, or $\exists T$ are $\Theta(N^{3/2})$ by use of worst-case optimal join algorithms

What about Cyclic Queries

- This approach proposed in:

Incremental View Maintenance with Triple Lock Factorization Benefits.
Milos Nikolic, Dan Olteanu:
SIGMOD Conference 2018: 365-380

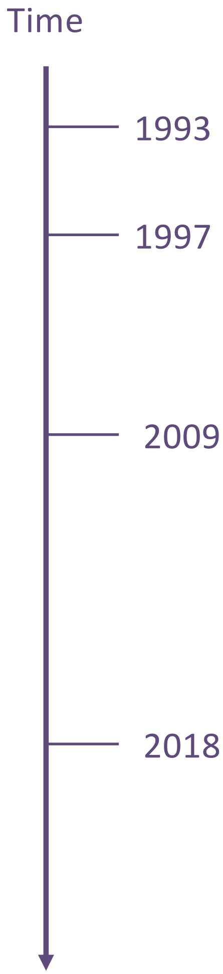
- F-IVM features:
  - View trees instead of GJTs for HIVM-based processing also allowing cyclic queries
  - Processing of complex aggregations (see later)
  - Exploiting factorized representations of updates and results (see later)

Main Algorithmic Ideas

1. IVM ≡ processing of delta queries

2. Materialize results of subqueries in addition to the actual query result

3. Exploit data skew



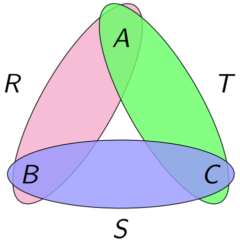
Exploiting Data Skew

- The maintenance approaches considered so far exploit query structure but not data skew.
- These approaches do not achieve worst-case optimal update and answer times in general.
  - Exception: q-hierarchical queries
- We present a maintenance approach that takes data skew into account and admits:
  - Worst-case optimal update and answer times
  - Time-space trade-off

Counting Triangles under Updates in Worst-Case Optimal Time.
Ahmet Kara, Hung Q. Ngo, Milos Nikolic, Dan Olteanu, Haozhe Zhang.
To appear in ICDT 2019

Example: The Triangle Count

Maintain the triangle count Q
under single-tuple updates to R, S, and T!



*Q counts the number of tuples
in the join of R, S, and T.*

$$Q = \pi_{(\,)}\big[R(a,b) \bowtie S(b,c) \bowtie T(c,a)\big]$$

Updates to the Triangle Count

R

A	B	
a_1	b_1	2
a_2	b_1	3

S

B	C	
b_1	c_1	2
b_1	c_2	1

T

C	A	
c_1	a_1	1
c_2	a_1	3
c_2	a_2	3

Updates to the Triangle Count

R

A	B	
a_1	b_1	2
a_2	b_1	3

S

B	C	
b_1	c_1	2
b_1	c_2	1

T

C	A	
c_1	a_1	1
c_2	a_1	3
c_2	a_2	3

$R \bowtie S \bowtie T$

A	B	C	
a_1	b_1	c_1	$2 \cdot 2 \cdot 1 = 4$

Updates to the Triangle Count

R

A	B	
a_1	b_1	2
a_2	b_1	3

S

B	C	
b_1	c_1	2
b_1	c_2	1

T

C	A	
c_1	a_1	1
c_2	a_1	3
c_2	a_2	3

$R \bowtie S \bowtie T$

A	B	C	
a_1	b_1	c_1	$2 \cdot 2 \cdot 1 = 4$
a_1	b_1	c_2	$2 \cdot 1 \cdot 3 = 6$
a_2	b_1	c_2	$3 \cdot 1 \cdot 3 = 9$

Updates to the Triangle Count

R

A	B	
a_1	b_1	2
a_2	b_1	3

S

B	C	
b_1	c_1	2
b_1	c_2	1

T

C	A	
c_1	a_1	1
c_2	a_1	3
c_2	a_2	3

$R \bowtie S \bowtie T$

A	B	C	
a_1	b_1	c_1	$2 \cdot 2 \cdot 1 = 4$
a_1	b_1	c_2	$2 \cdot 1 \cdot 3 = 6$
a_2	b_1	c_2	$3 \cdot 1 \cdot 3 = 9$

↓

$Q(\mathbf{D})$

$\emptyset$	
$(\,)$	$4 + 6 + 9 = 19$

Updates to the Triangle Count

R

A	B	
a_1	b_1	2
a_2	b_1	3

S

B	C	
b_1	c_1	2
b_1	c_2	1

T

C	A	
c_1	a_1	1
c_2	a_1	3
c_2	a_2	3

$R \bowtie S \bowtie T$

A	B	C	
a_1	b_1	c_1	$2 \cdot 2 \cdot 1 = 4$
a_1	b_1	c_2	$2 \cdot 1 \cdot 3 = 6$
a_2	b_1	c_2	$3 \cdot 1 \cdot 3 = 9$

$\uparrow$

$\Delta R(a, b)$

A	B	
a_2	b_1	-2

$\downarrow$

$Q(\mathbf{D})$

$\emptyset$	
$(\,)$	$4 + 6 + 9 = 19$

Updates to the Triangle Count

R

A	B	
a_1	b_1	2
a_2	b_1	3

S

B	C	
b_1	c_1	2
b_1	c_2	1

T

C	A	
c_1	a_1	1
c_2	a_1	3
c_2	a_2	3

$R \bowtie S \bowtie T$

A	B	C	
a_1	b_1	c_1	$2 \cdot 2 \cdot 1 = 4$
a_1	b_1	c_2	$2 \cdot 1 \cdot 3 = 6$
a_2	b_1	c_2	$3 \cdot 1 \cdot 3 = 9$

↑

$\Delta R(a, b)$

A	B	
a_2	b_1	-2

↓

$Q(\mathbf{D})$

$\emptyset$	
( )	$4 + 6 + 9 = 19$

Updates to the Triangle Count

R

A	B	
a_1	b_1	2
a_2	b_1	1

S

B	C	
b_1	c_1	2
b_1	c_2	1

T

C	A	
c_1	a_1	1
c_2	a_1	3
c_2	a_2	3

$R \bowtie S \bowtie T$

A	B	C	
a_1	b_1	c_1	$2 \cdot 2 \cdot 1 = 4$
a_1	b_1	c_2	$2 \cdot 1 \cdot 3 = 6$
a_2	b_1	c_2	$3 \cdot 1 \cdot 3 = 9$

↑

$\Delta R(a, b)$

A	B	
a_2	b_1	-2

↓

$Q(\mathbf{D})$

$\emptyset$	
$(\,)$	$4 + 6 + 9 = 19$

Updates to the Triangle Count

R

A	B	
a_1	b_1	2
a_2	b_1	1

S

B	C	
b_1	c_1	2
b_1	c_2	1

T

C	A	
c_1	a_1	1
c_2	a_1	3
c_2	a_2	3

$R \bowtie S \bowtie T$

A	B	C	
a_1	b_2	c_2	$2 \cdot 2 \cdot 1 = 4$
a_1	b_1	c_2	$2 \cdot 1 \cdot 3 = 6$
a_2	b_1	c_3	$3 \cdot 1 \cdot 3 = 9$

$\uparrow$

$\downarrow$

$\Delta R(a, b)$

A	B	
a_2	b_1	-2

$Q(\mathbf{D})$

$\emptyset$	
$(\,)$	$4 + 6 + 9 = 19$

Updates to the Triangle Count

R

A	B	
a_1	b_1	2
a_2	b_1	1

S

B	C	
b_1	c_1	2
b_1	c_2	1

T

C	A	
c_1	a_1	1
c_2	a_1	3
c_2	a_2	3

$R \bowtie S \bowtie T$

A	B	C	
a_1	b_2	c_2	$2 \cdot 2 \cdot 1 = 4$
a_1	b_1	c_2	$2 \cdot 1 \cdot 3 = 6$
a_2	b_1	c_3	$1 \cdot 1 \cdot 3 = 3$

↑

$\Delta R(a, b)$

A	B	
a_2	b_1	-2

↓

$Q(\mathbf{D})$

$\emptyset$	
( )	$4 + 6 + 9 = 19$

Updates to the Triangle Count

R

A	B	
a_1	b_1	2
a_2	b_1	1

S

B	C	
b_1	c_1	2
b_1	c_2	1

T

C	A	
c_1	a_1	1
c_2	a_1	3
c_2	a_2	3

$R \bowtie S \bowtie T$

A	B	C	
a_1	b_2	c_2	$2 \cdot 2 \cdot 1 = 4$
a_1	b_1	c_2	$2 \cdot 1 \cdot 3 = 6$
a_2	b_1	c_3	$1 \cdot 1 \cdot 3 = 3$

↑

$\Delta R(a, b)$

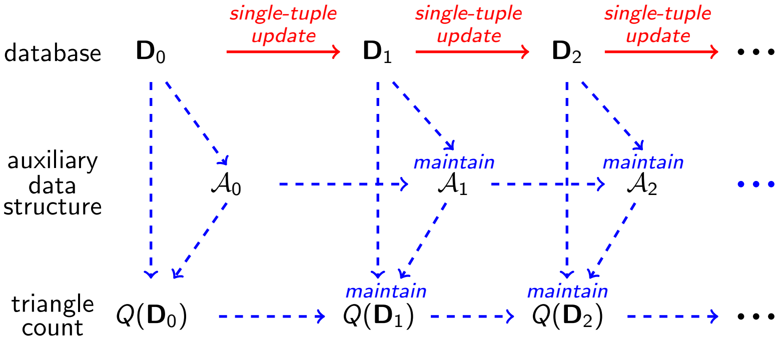
A	B	
a_2	b_1	-2

↓

$Q(\mathbf{D})$

$\emptyset$	
$(\,)$	$4 + 6 + 3 = 13$

The Considered Maintenance Problem



Given a current database $\mathbf{D}$ and a single-tuple update,
what are the time and space complexities for maintaining $Q(\mathbf{D})$?

Much Ado about Triangles

The Triangle Query Served as Milestone in Many Fields

- Worst-case optimal join algorithms *[Algorithmica 1997, SIGMOD R. 2013]*
- Parallel query evaluation *[Found. & Trends DB 2018]*
- Randomized approximation in static settings *[FOCS 2015]*
- Randomized approximation in data streams *[SODA 2002, COCOON 2005, PODS 2006, PODS 2016, Theor. Comput. Sci. 2017]*

Intensive Investigation of Answering Queries under Updates

- Theoretical developments *[PODS 2017, ICDT 2018]*
- Systems developments *[F. & T. DB 2012, VLDB J. 2014, SIGMOD 2017, 2018]*
- Lower bounds *[STOC 2015, ICM 2018]*

So far: **No** dynamic algorithm maintaining the **exact triangle count** in **worst-case optimal** time!

Naïve Maintenance

"Compute from scratch!"

$$\pi_{()}\left[\Big(\underbrace{R(a,b) + \Delta R(a,b)}_{newR}\Big) \bowtie S(b,c) \bowtie T(c,a)\right]$$

$$=$$

$$\pi_{()}\left[newR(a,b) \bowtie S(b,c) \bowtie T(c,a)\right]$$

Maintenance Complexity

- Time: $\mathcal{O}(|\mathbf{D}|^{1.5})$ using worst-case optimal join algorithms
- Space: $\mathcal{O}(|\mathbf{D}|)$ to store input relations

Classical IVM

"Compute the difference!"

Let $\Delta R(a,b) = \{(a',b') \mapsto m\}$

$$\pi_{()}\Big[\big(\,R(a,b) + \Delta R(a,b)\,\big) \bowtie S(b,c) \bowtie T(c,a)\Big]$$

$$=$$

$$\pi_{()}\Big[R(a,b) \bowtie S(b,c) \bowtie T(c,a)\Big]$$

$$+$$

$$\pi_{()}\Big[\Delta R(a,b) \bowtie \sigma_{b=b'} S(b,c) \bowtie \sigma_{a=a'} T(c,a)\Big]$$

Maintenance Complexity

- Time: $\mathcal{O}(|\mathbf{D}|)$ to intersect C-values from S and T
- Space: $\mathcal{O}(|\mathbf{D}|)$ to store input relations

Higher Order IVM

"Compute the difference by using pre-materialized views!"

Let $\Delta R(a,b) = \{(a',b') \mapsto m\}$

Pre-materialize $V_{ST}(b,a) = \pi_{b,a}\, S(b,c) \bowtie T(c,a)$!

$$\pi_{()}\Big[\big(\, R(a,b) + \Delta R(a,b)\,\big) \bowtie S(b,c) \bowtie T(c,a)\Big]$$
$$=$$
$$\pi_{()}\Big[R(a,b) \bowtie S(b,c) \bowtie T(c,a)\Big]$$
$$+$$
$$\pi_{()}\Big[\Delta R(a,b) \bowtie \sigma_{a=a',b=b'} V_{ST}(b,a)\Big]$$

Maintenance Complexity

- Time for updates to R: $\mathcal{O}(1)$ to look up in V_{ST}
- Time for updates to S and T: $\mathcal{O}(|\mathbf{D}|)$ to maintain V_{ST}
- Space: $\mathcal{O}(|\mathbf{D}|^2)$ to store input relations and V_{ST} (improvable to $\mathcal{O}(|\mathbf{D}|^{1.5})$)

Closing the Complexity Gap

Complexity bounds for the maintenance of the triangle count

Known Upper Bound

Maintenance Time: $\mathcal{O}(|\mathbf{D}|)$

Space: $\mathcal{O}(|\mathbf{D}|)$

Known Lower Bound

Amortized maintenance time: not $\mathcal{O}(|\mathbf{D}|^{0.5-\gamma})$ for any $\gamma > 0$

(under reasonable complexity theoretic assumptions)

Closing the Complexity Gap

Complexity bounds for the maintenance of the triangle count

Known Upper Bound

Maintenance Time: $\mathcal{O}(|\mathbf{D}|)$

Space: $\mathcal{O}(|\mathbf{D}|)$

Can the triangle count be maintained in sublinear time?

Known Lower Bound

Amortized maintenance time: not $\mathcal{O}(|\mathbf{D}|^{0.5-\gamma})$ for any $\gamma > 0$

(under reasonable complexity theoretic assumptions)

Closing the Complexity Gap

Complexity bounds for the maintenance of the triangle count

Known Upper Bound

Maintenance Time: $\mathcal{O}(|\mathbf{D}|)$

Space: $\mathcal{O}(|\mathbf{D}|)$

Can the triangle count be maintained in sublinear time?

Yes!

We propose: IVM$^{\varepsilon}$

Amortized maintenance time:

$\mathcal{O}(|\mathbf{D}|^{0.5})$

This is worst-case optimal!

Known Lower Bound

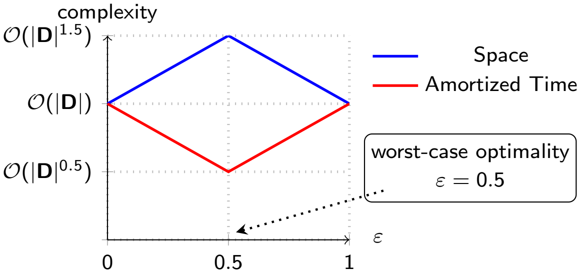
Amortized maintenance time: not $\mathcal{O}(|\mathbf{D}|^{0.5-\gamma})$ for any $\gamma > 0$

(under reasonable complexity theoretic assumptions)

IVM$^\varepsilon$ Exhibits a Time-Space Tradeoff

Given $\varepsilon \in [0,1]$, IVM$^\varepsilon$ maintains the triangle count with

- $\mathcal{O}(|\mathbf{D}|^{\max\{\varepsilon,1-\varepsilon\}})$ amortized time and
- $\mathcal{O}(|\mathbf{D}|^{1+\min\{\varepsilon,1-\varepsilon\}})$ space.



worst-case optimality
$\varepsilon = 0.5$

- Known maintenance approaches are recovered by IVM$^\varepsilon$.

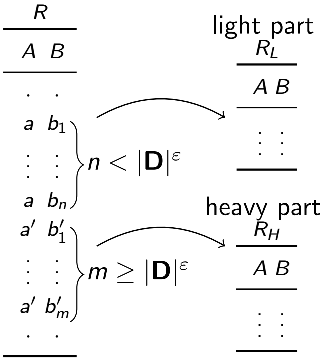
Main Ideas in IVM$^\varepsilon$

- Compute the difference like in classical IVM!
- Materialize views like in Higher Order IVM!
- New ingredient: Use adaptive processing based on data skew!
  $\Longrightarrow$ Treat *heavy* values differently from *light* values!

Relation Partitioning

Fix $\varepsilon \in [0,1]$ and partition R into

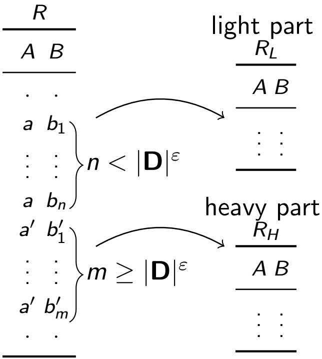
- a light part $R_L = \{t \in R \mid |\sigma_{A=t.A}| < |\mathbf{D}|^{\varepsilon}\}$,
- a heavy part $R_H = R \setminus R_L$!



Relation Partitioning

Fix $\varepsilon \in [0,1]$ and partition R into

- a light part $R_L = \{t \in R \mid |\sigma_{A=t.A}| < |\mathbf{D}|^{\varepsilon}\}$,
- a heavy part $R_H = R \backslash R_L$!



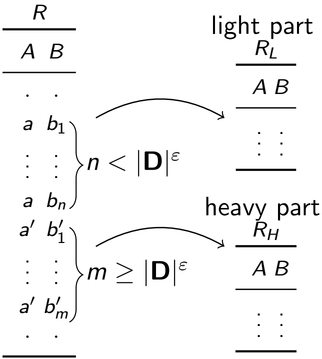
Derived Bounds

- for all A-values a: $|\sigma_{A=a} R_L| < |\mathbf{D}|^{\varepsilon}$
- $|\pi_A R_H| \leq |\mathbf{D}|^{1-\varepsilon}$

Relation Partitioning

Fix $\varepsilon \in [0,1]$ and partition R into

- a light part $R_L = \{t \in R \mid |\sigma_{A=t.A}| < |\mathbf{D}|^\varepsilon\}$,
- a heavy part $R_H = R \setminus R_L$!



R

$A \; B$

$a \; b_1$ … $a \; b_n$ $\Big\}\, n < |\mathbf{D}|^\varepsilon$

$a' \; b'_1$ … $a' \; b'_m$ $\Big\}\, m \geq |\mathbf{D}|^\varepsilon$

light part

R_L

$A \; B$

heavy part

R_H

$A \; B$

Derived Bounds

- for all A-values a: $|\sigma_{A=a} R_L| < |\mathbf{D}|^\varepsilon$
- $|\pi_A R_H| \leq |\mathbf{D}|^{1-\varepsilon}$

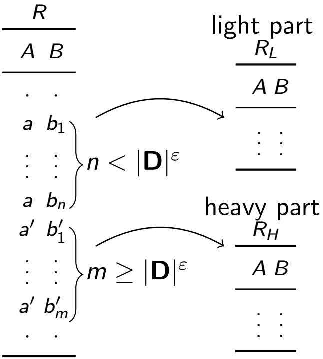
Likewise, partition

- $S = S_L \cup S_H$ based on B, and
- $T = T_L \cup T_H$ based on C!

Relation Partitioning

Fix $\varepsilon \in [0,1]$ and partition R into

- a light part $R_L = \{t \in R \mid |\sigma_{A=t.A}| < |\mathbf{D}|^{\varepsilon}\}$,
- a heavy part $R_H = R \backslash R_L$!



Derived Bounds

- for all A-values a: $|\sigma_{A=a} R_L| < |\mathbf{D}|^{\varepsilon}$
- $|\pi_A R_H| \leq |\mathbf{D}|^{1-\varepsilon}$

Likewise, partition

- $S = S_L \cup S_H$ based on B, and
- $T = T_L \cup T_H$ based on C!

Q is the sum of skew-aware views

$$\pi_{()}\left[R_U(a,b) \bowtie S_V(b,c) \bowtie T_W(c,a)\right]$$

with $U, V, W \in \{L, H\}$.

Adaptive Maintenance Strategy

Given an update $\Delta R_*(a, b) = \{(a', b') \mapsto m\}$, compute the difference for each skew-aware view using different strategies:

Skew-aware View	Evaluation from left to right	Time
$\pi_{()}\left[R_*(a, b) \bowtie S_L(b, c) \bowtie T_L(c, a)\right]$	$\Delta R_*(a', b') \cdot \sum_{c'} S_L(b', c') \cdot T_L(c', a')$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{\varepsilon})$

Adaptive Maintenance Strategy

Given an update $\Delta R_*(a, b) = \{(a', b') \mapsto m\}$, compute the difference for each skew-aware view using different strategies:

Skew-aware View	Evaluation from left to right	Time
$\pi_{(\,)}\left[R_*(a,b) \bowtie S_L(b,c) \bowtie T_L(c,a)\right]$	$\Delta R_*(a',b') \cdot \sum_{c'} S_L(b',c') \cdot T_L(c',a')$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{\varepsilon})$
$\pi_{(\,)}\left[R_*(a,b) \bowtie S_H(b,c) \bowtie T_H(c,a)\right]$	$\Delta R_*(a',b') \cdot \sum_{c'} T_H(c',a') \cdot S_H(b',c')$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{1-\varepsilon})$

Adaptive Maintenance Strategy

Given an update $\Delta R_*(a, b) = \{(a', b') \mapsto m\}$, compute the difference for each skew-aware view using different strategies:

Skew-aware View	Evaluation from left to right	Time
$\pi_{(\,)}\big[R_*(a,b) \bowtie S_L(b,c) \bowtie T_L(c,a)\big]$	$\Delta R_*(a',b') \cdot \sum_{c'} S_L(b',c') \cdot T_L(c',a')$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{\varepsilon})$
$\pi_{(\,)}\big[R_*(a,b) \bowtie S_H(b,c) \bowtie T_H(c,a)\big]$	$\Delta R_*(a',b') \cdot \sum_{c'} T_H(c',a') \cdot S_H(b',c')$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{1-\varepsilon})$
$\pi_{(\,)}\big[R_*(a,b) \bowtie S_L(b,c) \bowtie T_H(c,a)\big]$	$\Delta R_*(a',b') \cdot \sum_{c'} S_L(b',c') \cdot T_H(c',a')$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{\varepsilon})$
	or	
	$\Delta R_*(a',b') \cdot \sum_{c'} T_H(c',a') \cdot S_L(b',c')$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{1-\varepsilon})$

Adaptive Maintenance Strategy

Given an update $\Delta R_*(a, b) = \{(a', b') \mapsto m\}$, compute the difference for each skew-aware view using different strategies:

Skew-aware View	Evaluation from left to right	Time
$\pi_{()}\big[R_*(a,b) \bowtie S_L(b,c) \bowtie T_L(c,a)\big]$	$\Delta R_*(a',b') \cdot \sum_{c'} S_L(b',c') \cdot T_L(c',a')$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{\varepsilon})$
$\pi_{()}\big[R_*(a,b) \bowtie S_H(b,c) \bowtie T_H(c,a)\big]$	$\Delta R_*(a',b') \cdot \sum_{c'} T_H(c',a') \cdot S_H(b',c')$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{1-\varepsilon})$
$\pi_{()}\big[R_*(a,b) \bowtie S_L(b,c) \bowtie T_H(c,a)\big]$	$\Delta R_*(a',b') \cdot \sum_{c'} S_L(b',c') \cdot T_H(c',a')$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{\varepsilon})$
	or	
	$\Delta R_*(a',b') \cdot \sum_{c'} T_H(c',a') \cdot S_L(b',c')$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{1-\varepsilon})$
$\pi_{()}\big[R_*(a,b) \bowtie S_H(b,c) \bowtie T_L(c,a)\big]$	$\Delta R_*(a',b') \cdot V_{ST}(b',a')$	$\mathcal{O}(1)$

Adaptive Maintenance Strategy

Given an update $\Delta R_*(a, b) = \{(a', b') \mapsto m\}$, compute the difference for each skew-aware view using different strategies:

Skew-aware View	Evaluation from left to right	Time
$\pi_{(\,)}\big[R_*(a,b) \bowtie S_L(b,c) \bowtie T_L(c,a)\big]$	$\Delta R_*(a',b') \cdot \sum_{c'} S_L(b',c') \cdot T_L(c',a')$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{\varepsilon})$
$\pi_{(\,)}\big[R_*(a,b) \bowtie S_H(b,c) \bowtie T_H(c,a)\big]$	$\Delta R_*(a',b') \cdot \sum_{c'} T_H(c',a') \cdot S_H(b',c')$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{1-\varepsilon})$
	$\Delta R_*(a',b') \cdot \sum_{c'} S_L(b',c') \cdot T_H(c',a')$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{\varepsilon})$
$\pi_{(\,)}\big[R_*(a,b) \bowtie S_L(b,c) \bowtie T_H(c,a)\big]$	or	
	$\Delta R_*(a',b') \cdot \sum_{c'} T_H(c',a') \cdot S_L(b',c')$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{1-\varepsilon})$
$\pi_{(\,)}\big[R_*(a,b) \bowtie S_H(b,c) \bowtie T_L(c,a)\big]$	$\Delta R_*(a',b') \cdot V_{ST}(b',a')$	$\mathcal{O}(1)$

Overall update time: $\mathcal{O}(|\mathbf{D}|^{\max(\varepsilon,1-\varepsilon)})$

Materialized Auxiliary Views

$$V_{RS}(a,c) = \pi_{a,c}\left[R_H(a,b) \bowtie S_L(b,c)\right]$$
$$V_{ST}(b,a) = \pi_{b,a}\left[S_H(b,c) \bowtie T_L(c,a)\right]$$
$$V_{TR}(a,c) = \pi_{a,c}\left[T_H(c,a) \bowtie R_L(a,b)\right]$$

- Maintenance of $V_{RS}(a,c) = \pi_{a,c}\left[R_H(a,b) \bowtie S_L(b,c)\right]$

Update	Evaluation from left to right	Time
$\Delta R_H(a,b) = \{(a',b') \mapsto m\}$	$\Delta R_H(a',b') \cdot \sum_{c'} S_L(b',c')$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{\varepsilon})$
$\Delta S_L(b,c) = \{(b',c') \mapsto m\}$	$\Delta S_L(b',c') \cdot \sum_{a'} R_H(a',b')$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{1-\varepsilon})$

Materialized Auxiliary Views

$$\begin{aligned} V_{RS}(a,c) &= \pi_{a,c}\left[R_H(a,b) \bowtie S_L(b,c)\right] \\ V_{ST}(b,a) &= \pi_{b,a}\left[S_H(b,c) \bowtie T_L(c,a)\right] \\ V_{TR}(a,c) &= \pi_{a,c}\left[T_H(c,a) \bowtie R_L(a,b)\right] \end{aligned}$$

- Maintenance of $V_{RS}(a,c) = \pi_{a,c}\left[R_H(a,b) \bowtie S_L(b,c)\right]$

Update	Evaluation from left to right	Time
$\Delta R_H(a,b) = \{(a',b') \mapsto m\}$	$\Delta R_H(a',b') \cdot \sum_{c'} S_L(b',c')$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{\varepsilon})$
$\Delta S_L(b,c) = \{(b',c') \mapsto m\}$	$\Delta S_L(b',c') \cdot \sum_{a'} R_H(a',b')$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{1-\varepsilon})$

- Size of $V_{RS}(a,c) = \pi_{a,c}\left[R_H(a,b) \bowtie S_L(b,c)\right]$

$$\begin{aligned} |V_{RS}(a,c)| &\leq |R_H| \cdot \max_{b'}\{|\sigma_{b=b'} S_L(b,c)|\} = \mathcal{O}(|\mathbf{D}|^{1+\varepsilon}) \\ |V_{RS}(a,c)| &\leq |S_L| \cdot \max_{b'}\{|\sigma_{b=b'} R_H(a,b)|\} = \mathcal{O}(|\mathbf{D}|^{1+(1-\varepsilon)}) \end{aligned}$$

Materialized Auxiliary Views

$$\begin{aligned} V_{RS}(a,c) &= \pi_{a,c}\left[R_H(a,b) \bowtie S_L(b,c)\right] \\ V_{ST}(b,a) &= \pi_{b,a}\left[S_H(b,c) \bowtie T_L(c,a)\right] \\ V_{TR}(a,c) &= \pi_{a,c}\left[T_H(c,a) \bowtie R_L(a,b)\right] \end{aligned}$$

- Maintenance of $V_{RS}(a,c) = \pi_{a,c}\left[R_H(a,b) \bowtie S_L(b,c)\right]$

Update	Evaluation from left to right	Time
$\Delta R_H(a,b) = \{(a',b') \mapsto m\}$	$\Delta R_H(a',b') \cdot \sum_{c'} S_L(b',c')$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{\varepsilon})$
$\Delta S_L(b,c) = \{(b',c') \mapsto m\}$	$\Delta S_L(b',c') \cdot \sum_{a'} R_H(a',b')$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{1-\varepsilon})$

- Size of $V_{RS}(a,c) = \pi_{a,c}\left[R_H(a,b) \bowtie S_L(b,c)\right]$

$$\begin{aligned} \lvert V_{RS}(a,c)\rvert &\leq \lvert R_H\rvert \cdot \max_{b'}\{\lvert\sigma_{b=b'} S_L(b,c)\rvert\} = \mathcal{O}(\lvert\mathbf{D}\rvert^{1+\varepsilon}) \\ \lvert V_{RS}(a,c)\rvert &\leq \lvert S_L\rvert \cdot \max_{b'}\{\lvert\sigma_{b=b'} R_H(a,b)\rvert\} = \mathcal{O}(\lvert\mathbf{D}\rvert^{1+(1-\varepsilon)}) \end{aligned}$$

- Overall: Update Time $\mathcal{O}(\lvert\mathbf{D}\rvert^{\max\{\varepsilon,1-\varepsilon\}})$; Space $\mathcal{O}(\lvert\mathbf{D}\rvert^{1+\min\{\varepsilon,1-\varepsilon\}})$

Rebalancing Partitions

- Updates can change the frequencies of values in the relation parts!
- This can require rebalancing of partitions.

  $\Longrightarrow$ Minor rebalancing: Transfer tuples from one to the other part of the same relation!

  $\Longrightarrow$ Major rebalancing: Recompute partitions and views from scratch!
- Both forms of rebalancing require superlinear time.
- The rebalancing times amortize over sequences of updates.

Extensions of IVM$^\varepsilon$?

Generalization of IVM$^\varepsilon$

- Partitioning on both attributes of each relation improves space complexity.
- IVM$^\varepsilon$ variants obtain worst-case optimal maintenance time for counting versions of Loomis-Whitney, 4-cycle, and 4-path.

Ongoing Work

- Characterization of the class of conjunctive count queries that admit $\mathcal{O}(\mathbf{D}^{0.5})$ worst-case optimal maintenance time
- Implementation of IVM$^\varepsilon$